Dynamic Feature Acquisition and Classification with Transformers in a Reinforcement Learning Framework

Gabriel Baruque
Department of Electrical Engineering
Pontifícia Universidade Católica
Rio de Janeiro, Brazil
gabriel@baruque.com.br

Wouter Caarls
Department of Electrical Engineering
Pontifícia Universidade Católica
Rio de Janeiro, Brazil
wouter@ele.puc-rio.br

***Abstract*—Classification with Costly Features (CwCF) addresses the challenge of balancing feature acquisition costs with classification accuracy. Traditional methods often rely on static feature subsets, leading to suboptimal resource utilization. This paper proposes a novel reinforcement learning (RL) framework enhanced with Transformer architecture to dynamically select features in a sequential manner, optimizing cost-accuracy trade-offs. Our model leverages the Transformer architecture to handle variable-length input sequences, combining embeddings for both feature values and their respective feature identities, while integrating Deep Q-Networks for decision-making. Experiments across synthetic and real-world benchmarks demonstrate that the Transformer-based approach achieves competitive accuracy compared to prior RL methods, though it falls short of state-of-the-art (SotA) models in cumulative return. The results highlight the model's ability to adaptively gather informative features, emphasizing accuracy over cost minimization. This work brings out the potential of sequence-based architectures in dynamic feature selection and opens discussion for further exploration of Transformer-RL hybrids in cost-sensitive classification tasks.**

***Index Terms*—Costly Features, Dynamic Feature Acquisition, Transformers, Classification, Reinforcement Learning**

I. Introduction

Despite recent advances in the storage, processing, and consumption of structured and unstructured data, tabular data remains the most widely used format in real-world applications [1], including healthcare [2], time-series analysis [3], and reinforcement learning [4]. Similarly, classification tasks, though well-established, continue to pose unresolved challenges.

In real-world classification tasks, acquiring features often incurs costs (e.g. time, money), where the total resource expenditure equals the sum of individual feature costs. In Machine Learning (ML) this scenario is called Budgeted ML, and the final objective demands efficient resource allocation while maintaining high accuracy. This can be seen in different applications, such as Medical diagnosis [5] and text classification [6]. When this problem is specifically posed in a tabular dataset setup, it can be called Classification with Costly Features (CwCF).

Research funded by CNPq (project number 140059/2020-2), and PUC.

Traditional methods often rely on static feature selection, where the same subset of features is used for all samples. This inflexibility leads to suboptimal resource utilization, as critical features for one sample may be irrelevant for another [7].

To address this, we propose a reinforcement learning (RL)-based approach in which an agent dynamically acquires features sequentially, stopping when sufficient information is gathered for confident classification. Our model integrates Deep Q-Networks (DQNs) [8] with supervised learning to optimize the trade-off between cost and accuracy.

Our work, derived from [9], extends previous research by leveraging the power of Transformers [10], which have achieved impressive results in different applications [11]–[13]. We propose a Transformer-based model used in an RL framework while maintaining the previous ideas of dynamic feature selection embedded with classification and combining supervised learning-like strategies within an RL approach.

The paper is organized as follows. Section II reviews related work on cost-sensitive classification and RL. Section III details the background concepts. Section IV presents our methodology. Section V evaluates performance on benchmark datasets, and Section VI concludes with future directions.

II. Related Work

The challenge of Cost-sensitive Classification with Costly Features (CwCF) has been addressed through diverse methodologies, ranging from reinforcement learning (RL) to bio-inspired optimization. Early RL-based approaches framed the problem as a Markov Decision Process (MDP), where states encode acquired features and actions select features or predict classes. For instance, [14] pioneered this paradigm using Q-learning, while [15] later integrated Deep Q-Networks (DQNs) with an external classifier, jointly training the RL agent and classifier using loss-based rewards. Subsequent work refined this framework: [16] introduced online classification with feature cost estimation via certainty thresholds, and [17] enhanced flexibility by incorporating explicit budget constraints. Notably, [18] further extended their model to handle missing

features, achieving state-of-the-art performance. Critically, all prior RL methods rely on an external classifier, a dependency our work eliminates.

Beyond RL, tree-based methods have also been explored. In [19] authors proposed pruning random forests to balance accuracy and feature costs, later advancing this with adaptive gating models for tree-based classifiers [20]. These methods prioritize interpretability but lack the dynamic adaptability of RL frameworks.

Besides cost-sensitive classification, an important part of this research is the usage of Transformers in an RL framework. On this regard, some works explored this idea, usually applying the combination in problems focused in game-play agents and robot control tasks.

The Decision Transformer (DT) [21] abstracts RL as a sequence modeling problem, tackling the problem in an offline RL manner using Transformers. It casts the problem of RL as a conditional sequence modeling, conditioning an autoregressive model on the desired return, past states and actions. The model is capable of adjusting the autoregressive output to match the desired return. Actions, rewards and states are fed as sequences, and each modality goes through a dedicated embedding layer. A positional embedding is also used. It achieved state-of-the-art performance when compared with baseline models such as Conservative Q-Learning (CQL) [22], REM [23] and QR-DQN [24]

Similar to DT, the Trajectory Transformer [25] also addresses the RL problem as a sequence modeling problem, producing a sequence of actions that leads to the highest return possible. Despite its similarity with DT, it expect inputs as a sequence of states dimensions, actions dimensions and rewards subsequences (autoregressively discretized). Results how competitive or better results with baselines such as CQL and DT.

Different from previous models, the Q-Transformer [26] can be considered an offline RL model. The main contribution is the per action-dimension tokenization of Q-values. In this model, continuous actions are discretized in bins, and each bin Q-value is updated during training. Q-targets are based on the maximization of the next action dimension within the same timestep. For the last action dimension, the Q-value considers the discounted maximization of the first action dimension in the next timestep, and reward. Results achieve state-of-the-art in offline datasets of control tasks when compared to baselines such DT and Implicit Q-learning [27].

III. Background

A. Reinforcement Learning

We base the proposed approach mainly on two ML paradigms, RL and Transformers. The former (RL) is chosen due to how flexible it is to model the problems using reward functions, and how it suits sequential problems really well, usually through Markov Decision Process (MDP) [28]. Generally, RL algorithms train an agent to learn the optimal behavior through iterative interactions with an environment. The agents interacts with the environment based on the current state, a set of observable informations in the agent perspective, at each timestep, making a decision about which action it should take. Each action provides a reward, a quantitative feedback from the environment. The goal of the agent is to maximize the cumulative reward over time, learning a policy, a mapping from states to actions.

The proposed model is based on Deep Q-Networks (DQN) [8], [29]. DQN models use deep neural networks to approximate the Q-function, $Q(s, a)$, in high-dimensional state spaces. The function Q represents the expected cumulative reward of taking action a in state s. The usage of an experience replay played an important part in DQN success, allowing the storage of past transitions (s, a, r, s') and randomly sampling during training to break temporal correlations to stabilize learning. On top of that, a Target Network is used to compute target Q-values. This network is updated periodically to mitigate risk of divergence.

DQN received other improvements, such as Double DQN [30], which deals with the Q-value overestimation problem (due to max operator) by decoupling action selection and evaluation. In this approach, the main network with parameters θ is responsible for choosing actions, while the target network with parameters θ' is responsible for estimating their values, reducing overoptimism and improving policy stability. The learning target can be defined as

$$Y_t^{\mathrm{DDQN}} = r_{t+1} + \gamma Q\left(s_{t+1}, \arg\max_{a'} Q(s_{t+1}, a'; \theta); \theta^-\right). \tag{1}$$

with r representing reward, and γ representing discount factor.

Dueling DQN [31] is another improvement, where the Q-Network is divided in two streams: the value stream estimates the state intrinsic value, $V(s)$, and the Advantage stream estimates the relative importance of each action, $A(s, a)$. The final Q-value combines both estimates as $Q(s, a) = V(s) + A(s, a) - \mathrm{mean}(A(s, a))$. This allows the agent to learn robust policies even in the presence of actions with minimal impact on outcomes.

B. Transformers

Transformers were first introduced in [10], and originally designed to deal with machine translation in Natural Language Processing (NLP). Later, many other problems were tackled using this approach, achieving great performance, such as in language modelling [32], [33] and computer vision [11], [34].

The basic structure of Transformers is composed of three blocks: encoder, decoder, and attention blocks. Additionally, input information passes through an embedding layer, which maps input values to a fixed length state space, the embedding dimension. Different embeddings can be applied to extract relevant information from the input array and be merged (e.g., concatenation or addition), generating a sequential set of information called tokens, which will be attended to in the attention blocks.

Embeddings: Each input token is mapped to a dense vector of dimension d_{emb}, enabling richer representations. Categorical features use a learned lookup table, while continuous features

can be embedded via a trainable linear layer. Positional embeddings, commonly used, add information about token order [32].

Attention: Given query (Q), key (K), and value (V) vectors, attention computes scaled dot-product scores followed by softmax to weight relevant tokens. Multi-head attention runs this in parallel subspaces, capturing diverse dependency patterns [35].

Encoder: Process the embedded input sequence, using self-attention, transforming it into context-rich representations for subsequent use.

Decoder: Crucial for auto-regressive or sequence-generation tasks, it receives information from the encoder, incorporating masked self-attention for auto-regressive generation, and encoder-decoder cross-attention [36].

Masks: Used to standardize input length and impose auto-regressiveness in decoders during training. Padding masks: prevent attention to padding positions, accommodating variable-length inputs. Auto-regressive (look-ahead) masks: prevent attention to future tokens, enforcing autoregressiveness in decoder training.

IV. Methodology

The usage of a transformer architecture in a RL framework introduces scalability and a refined representation, allowing inputs to be structured as sequences with varying length, representing many transitions instead of only one (like standard RL techniques), and improving information retrieval with embedding layers. With that, the model can learn more complex information about the given context.

We tackle the problem of CwCF, first by modeling it with an MDP. Basically, the objective is to train an agent capable of balancing between measuring one extra feature or classifying with the information it gathered so far, considering feature costs. This decision, made at each timestep, is dynamic with respect to measured values, and can be thought as a type of dynamic feature selection.

A. MDP

The MDP is modeled as follows. Considering a dataset $\mathcal{D}$ with F features and C classes, the array $x = \{x_1, x_2, \ldots, x_{\mathrm{F}}\}$ represents one sample from $\mathcal{D}$, and $y \in [1, C]$ represents its true class. The tuple (x, y) comprises one environment for the proposed agent.

States are composed by vector $v = \{v_1, v_2, \ldots, v_t\}$, indicating values, and vector $f = \{f_1, f_2, \ldots, f_t\}$, indicating features. The size of both vectors are dynamic, and rises whenever the agent chooses to gather another feature. Initially, no information is given to the agent, therefore we denote v_0 and f_0 as initial state, with the former being an empty vector and the latter receiving one element representing the "no-feature" state.

Actions are divided in *measurement actions* $\mathcal{A}_{\mathrm{m}}$, and *classification actions* $\mathcal{A}_{\mathrm{c}}$. A measurement action a_{m}, $1 \leq \mathrm{m} \leq \mathrm{F}$, indicates the next feature to be measured, pointing to how the state vectors are going to be updated for the next timestep.

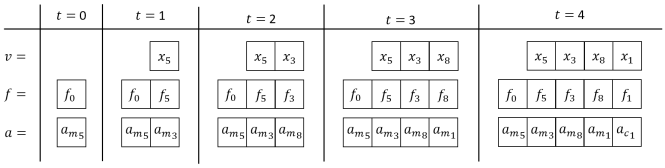


Fig. 1: Full episode example. All vectors increase dimension with each measurement action. This is only allowed because transformers can deal with varying length inputs.

Classification actions, a_{c}, $1 \leq \mathrm{c} \leq C$, are used to indicate the class that the information gathered so far should be assigned to. Also, classification actions are terminal. With that, the full set of actions has $F + C$ possible choices.

The transition function maps an action a to an outcome, usually updating the current state. In this implementation, the transition function has two outcomes based on which kind of action was chosen, following

$$T(v, f, a) = \begin{cases} (v', f') & , \text{if } a_{\mathrm{m}} \\ \text{terminal} & , \text{if } a_{\mathrm{c}} \end{cases} \tag{2}$$

where $v' = \text{append}(v, x_{\mathrm{m}}|a_{\mathrm{m}})$, and $f' = \text{append}(f, \mathrm{m}|a_{\mathrm{m}})$ are the next state vectors.

The reward function is based on the action types. Considering the cost of each feature as $\mathbf{c}_{\mathrm{f}}$, $1 \leq \mathrm{f} \leq \mathrm{F}$, measurement actions reflects the cost of each feature in the reward function. Classification actions, as they represent the actual classification, rewards with a positive value if the classification is correct, and a negative value otherwise. It is formalized as

$$R(s, a) = \begin{cases} -\mathbf{c}_{\mathrm{f}} & , \text{if } a_{\mathrm{m}} \\ 1 & , \text{if } a_{\mathrm{c}} = y \\ -1 & , \text{if } a_{\mathrm{c}} \neq y \end{cases}. \tag{3}$$

To facilitate understanding, a full episode is exemplified in Fig. 1.

B. Architecture

Although the network architecture is based on GPT [33], we implement an encoder-only strategy, allowing all inputs to communicate with each other during the attention mechanism. The model is composed of embedding blocks, multi-head self-attention blocks and two final heads, one for measurement actions and another for classification actions. Fig. 2 depicts the described architecture.

1) Embeddings: We split information into two different embeddings: values and features. To embed values coming from the v vector, we apply a linear layer with dimension $d_{\text{emb}} = 128$. Since the feature vector contains discrete values, f is embedded with a lookup table, also with size $d_{\text{emb}} = 128$. With that, both embedding layers output a vector in $\mathbb{R}^{d_{emb}}$, increasing the dimension from each value in the input sequence, and its respective feature (both in $\mathbb{R}^1$). The linear layer applies a *tanh* activation function, while the lookup table just associates an input value with a fixed vector.

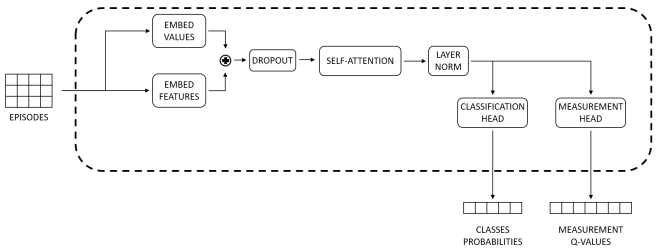


Fig. 2: Model's architecture, with dedicated embedding blocks for values and features, and two output heads, one for classification actions and another for measurement actions.

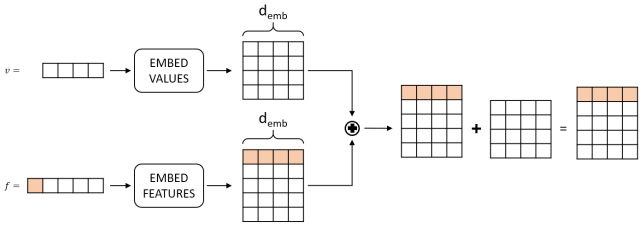


Fig. 3: Embedding blocks. The "no-feature" embedding is kept the same during embeddings addition.

As mentioned in IV-A, both state vectors starts differently. At timestep 0, $v = \{\}$ and $f = \{f_0\}$. This leads to different shaped vectors, with v having dimensions $(batch, context, d_{\text{emb}})$, and f having dimensions $(batch, 1 + context, d_{\text{emb}})$. both vectors are summed together to incorporate information in one vector, except the extra element in feature vector, which is left as is. This process is represented in Fig. 3.

C. Training process

During training, the well-known ε-greedy strategy was used to control exploration and exploitation of the action space, following Equation 4. As usual in RL models, at the start of the training process we set $\varepsilon = 1$, which is exponentially decayed on each training episode, until it reaches $\varepsilon = 0.1$. With that value, the model exploits actions that leads to good rewards most of the time, but still has room for exploration with random actions.

$$\pi_\varepsilon(a|s) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|\mathcal{A}|} & \text{if } a = \arg\max_{a'} Q(s, a') \\ \frac{\varepsilon}{|\mathcal{A}|} & \text{otherwise} \end{cases} \tag{4}$$

Standard algorithms, such as DQN, usually store transition tuples, such as (s, a, r, s'), to sample from during training. Since the reward is directly based on actions in a deterministic transition function, in our case it is enough to store the ground truth tuple (x, y) and the sequence of actions taken a. With that, all episode information can be rebuilt in the correct sequence.

At the beginning of the training, the model populates the memory by choosing actions randomly, until there are enough episodes to sample from. After that, the training strategy starts, basically divided in three iterative steps: populate memory, sample memory, training step.

In the first step, the model applies the ε-greedy strategy and updates the memory with more up-to-date episodes, starting more randomly, and gradually becoming more greedy. This keeps the samples updated to reflect the current policy, while still accounting for older behavior.

In the second step, the memory is sampled, and a batch of random episodes is extracted. Each sample stores the feature vector f, and the ground-truth tuple (x, y). With that, the value vector v can be restored and all information for training is available. The batch is truncated with a random amount of information, to simulate different sizes of inputs.

With that, the third step takes place, where the model trains over the processed batch. The attention mechanism is applied, however no masks are used, allowing all queries and keys to communicate with each other, ensuring context information is learned in all scenarios, from small to full contexts. During this step, as there are two final heads, a loss function that covers both objectives is needed, therefore Mean-Squared Error (MSE) is used for measurement Q-values, while Cross-Entropy (CE) is used for classification actions, following

$$\mathcal{L}(Y, \hat{Q}, y, \hat{y}) = \text{MSE}_{\text{measurement}} + \text{CE}_{\text{classification}} \tag{5}$$

. Specifically,

$$\text{MSE}_{\text{measurement}} = (\hat{Q} - Y)^2 \tag{6}$$

, where $\hat{Q}$ is the model predicted Q-value and Y is the updated target Q-value, and

$$\text{CE}_{\text{classification}} = -\sum_{c=1}^{C} \mathbb{I}_y \log(P(\hat{y})) \tag{7}$$

, where $\mathbb{I}_y$ is the indicator vector specifying if the class element c is the correct class, and $P(\hat{y})$ is the predicted probability distribution between classes.

It is noteworthy that in this implementation, target updates are performed both for all classification actions and for the chosen measurement action in one step. Achieving all actions updates, like in [9] which inspired the current update strategy, becomes unmanageable in this scenario, since all transformer outputs based on combinations of current and next states would need to be read out.

In an RL framework, the greedy action is chosen based on the highest Q-value. Since classification actions are trained based on CE, we need to convert the generated logits into Q-values, so that measurement actions and classification actions can be compared. We do that by first applying a softmax function to the classification head logits, and converting the generated probability distribution $P(\hat{y}_c)$ to the equivalent Q-value for each classification action c. This Q-value reflects the expected reward from Eq. 3 given the estimated successful

TABLE I: Data sets Characteristics

	Features	Classes	Samples	Training episodes
Wine	13	3	178	50k
Miniboone	50	2	130065	300k
Healthrisk	15	3	702	50k
Pen Digits	16	10	10992	50k
Beans	16	7	13611	50k

classification probability:

$$\begin{aligned} Q_{\text{equivalent}} &= 1P(\hat{y_c}) - 1(1 - P(\hat{y_c})) \\ &= 2P(\hat{y_c}) - 1 \end{aligned} . \tag{8}$$

In addition, two inductive biases are applied to the model. First, repeated actions are not allowed by the environment. This limitation is added due to the fact that information-wise, repeated actions provide only already known information, while reducing effective return. Second, during training the model is forced to always choose measurement actions as long as there are still measurements available. Since classification actions related targets are always updated and considered during the training step, the model benefits from longer episodes, with more measurement actions.

V. Experiments

A set of experiments, based on [9], was carried out in different datasets. Each experiment on a dataset comprises ten different runs with different splits for training, validation and test. The results reported in this section are an average from these ten runs. Experiments were carried out in the datasets: 'Synthetic', 'Wine' [37], 'Healthrisk' [38], 'Miniboone' [39], 'Pen Digits' [40] and 'Beans' [41]. Their characteristics are shown in Table I.

Feature costs are fixed to $\mathbf{c}_f = 0.03$, without lack of generality. Classification action rewards are set to 1 and -1 for correct and incorrect classifications, respectively. The hyperparameters of the proposed algorithm are initially based on the Decision Transformer [21] implementation, and were manually tuned from there to ensure a fair comparison with the baselines, which were also manually tuned.

Performance is assessed based on accuracy and return, and results are compared to the baselines proposed in [18], and [9], which were run with the same splits and equivalent costs for all datasets.

All experiments conducted were performed on a Linux platform with the following specifications: Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz, 256GB RAM, and a GTX 1080 TI. The programming language used was Python 3.7.9, with Pytorch 1.12.0.

A. Proof of Concept

Before testing on real datasets, we executed an experiment on the Synthetic dataset to assess the behavior of the model, and confirm that in fact it achieves a dynamic feature selection and classification. This simple dataset, shown in Fig. 4, is created with two features and classes, each generated by a normal distribution.

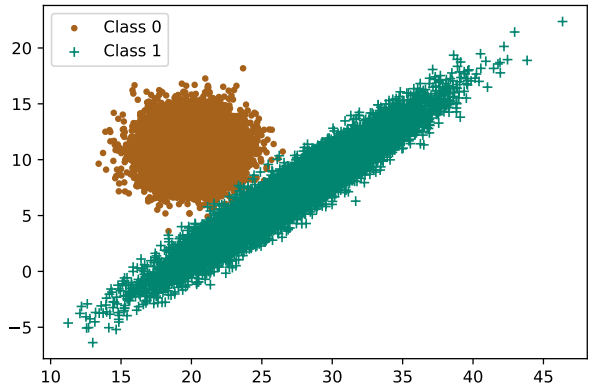


Fig. 4: The Synthetic dataset, with two features and two classes. The x-axis represents feature 0, and the y-axis represents feature 1.

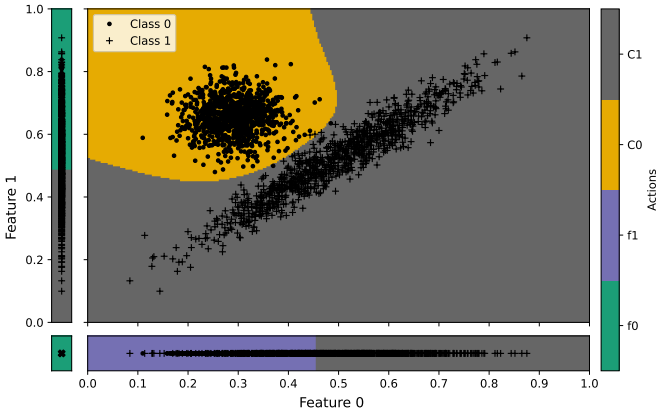


Fig. 5: Policy achieved on Synthetic dataset.The model achieves dynamic feature selection and classification. It classifies early when there is enough information, and measures an extra feature when needed.

After training the proposed model, we extract its policy in all possible states, as Fig. 5 depicts. The image has four states: initial state (lower-left), feature 0 measurement (bottom-right), feature 1 measurement (upper-left) and both features measurement (upper-right). In each state, the colors represent the action chosen by the model. Action "f0" means "measure feature 0", similarly for action "f1". Action "C0" means "classify as class 0", same for action "c1". We can observe how the model is able to classify with partial information, and ask for more information when it needs to.

We also assess the maximum Q-value obtained by the model, which shows how "certain" the model is about the chosen action. Fig. 6 shows the Q-values obtained at each state. We observe that the model has a high Q-value most of the time, with a slight decrease in regions with a mixture of different classes.

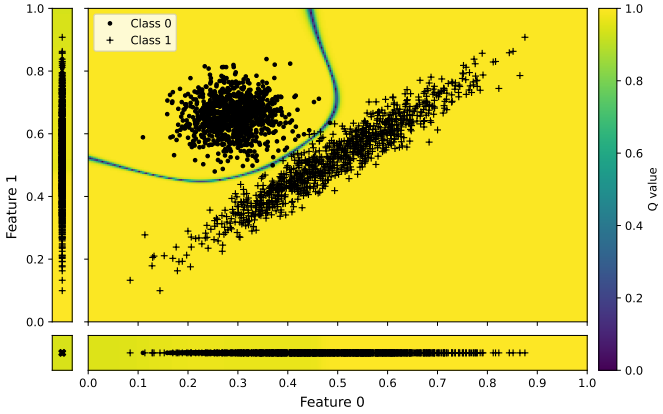


Fig. 6: Maximum Q-values on Synthetic dataset. It shows a thin decision boundary and a higly confident model.

B. Results

After assessing the Synthetic dataset, we move on to perform experiments on real datasets. As an extension of [9], the proposed model is compared with three main baselines. The first baseline is based on a standard approach of feature selection (using Recursive Feature Elimination) with a subsequent classification based on the best result between Logistic Regression and Linear Discriminant Analysis (same as in [9]), and is represented as "RFE + classifier". The second baseline is the model proposed in [9], which is based solely on Deep Reinforcement Learning with target updates on all Q-values at a specific state, and we represent it as "All Action Updates". The third baseline is the model proposed on (J. Janisch, T. Pevný, and V. Lisý - 2020) [18], the current state-of-the-art, represented as "SotA". For comparison, a feedforward classifier with 400 and 300 hidden units was trained on all features, yielding the "All Features" baseline.

1) Validation: Before exposing the results for test datasets, we show the validation return performance comparison throughout training between the "RFE + classifier" and "All Action Updates" baselines, and the current proposed model. Fig. 7 shows that the proposed model return performances are on par or higher in some datasets (i.e. Synthetic, Wine and Pen Digits), while lower in others (Healthrisk, Miniboone and Beans). Accuracy-wise, the proposed model achieved better results in four datasets (Wine, Miniboone, Pen Digits, and Beans).

The differences in training episodes are due to the slower convergence of the transformer model, which needed more training steps to achieve convergence. Transformer models are known to require lots of training examples, which is verified in these experiments.

2) Test: The following section presents the results obtained on the test dataset. Tables II and III respectively show the return and accuracy performances of all baselines compared to the proposed model. Test results followed the ones achieved during validation, with similar or better returns in half of the datasets and better accuracy in four, when comparing against the "All Action Updates" model. The SotA model generally achieved better performance both in return and accuracy.

To clarify, differently from the proposed model results that considers 10 runs with different seeds, the results about the "All Action Updates" model were obtained based on a 40 runs experiment with different seeds, therefore even though the former may produce wider confidence intervals, that does not necessarily imply less stability.

Our experiments show that integrating a Transformer into the RL framework yields competitive accuracy compared to the "All Action Updates" baseline, at the expense of somewhat lower cumulative returns in some environments. This trade-off suggests that the model learns more conservative measurement policies—favoring fewer risky misclassifications—while foregoing certain high-reward but riskier actions. Similar behavior has been observed in sequence-modeling approaches to RL, where the sequence model's inductive bias can dampen extreme action preferences [21], [42].

We conjecture that the strength of the proposed model would be more apparent in scenarios with variable costs between features. Also, it has been observed that standard transformer models, such as the one used, are unstable to train in RL frameworks. Applying architectures already validated in this scenario [42] may improve performance.

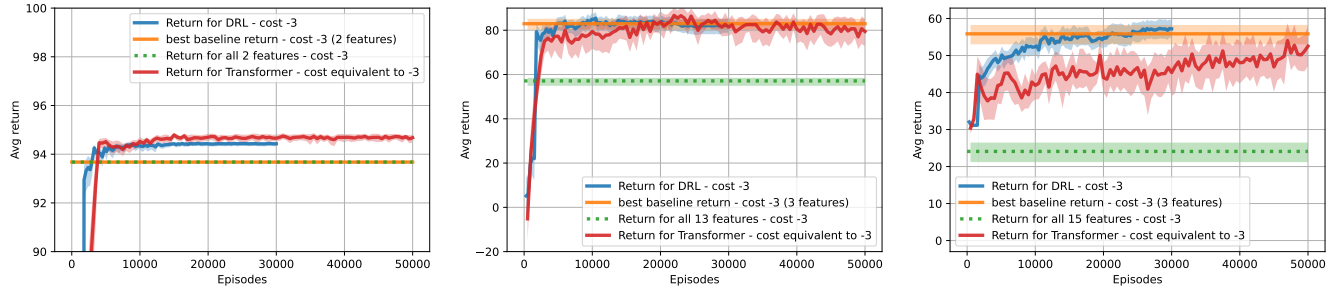
VI. Conclusion

In this paper we tackle the problem of CwCF with a novel strategy using Transformer models embedded in a RL framework. Because the Transformer architecture is suited to deal with inputs of varying lengths, unmeasured features have a natural way to be represented. Also, two embedding layers are applied, one for value, another for feature, improving the model's state representation.

Results obtained shows that, when compared with a previous implementation for this specific task [9], the proposed model achieves similar performance return-wise, and overall better performance accuracy-wise. However, it falls short on both when compared with the SotA.

Considering the results, the proposed Transformer model looks somewhat more cautious about classification than the "All Action Updates" model, since it achieves better overall accuracy while giving up on some return. Adjusting the loss function to give more importance to measurement actions might improve the performance in terms of return. Another possible approach is changing the discount factor, to prioritize more immediate rewards. Higher costs certainly would influence the model to measure less features, however it is still unclear why models achieved different outcomes with the same problem configuration.

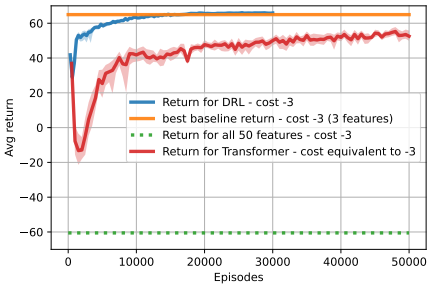
For next steps, experimenting with loss normalization, to ensure factors are at the same scale, and weighted loss, to control the importance of classification actions versus measurement actions, would be a natural extension. Moreover, adopting an auto-regressive approach during training, with a decoder-based architecture, is a promising alternative given the performances of current decoder-only models [33], [43].



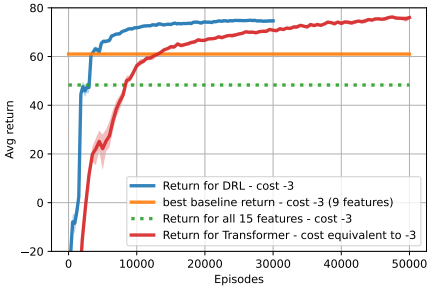
(a) Average returns on Synthetic dataset

(b) Average returns on 'Wine' dataset

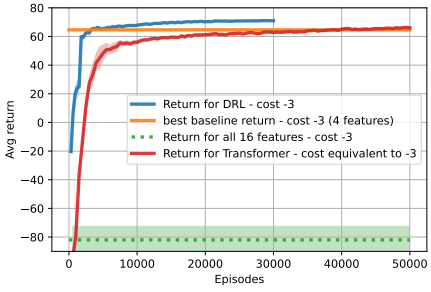
(c) Average returns on 'Healthrisk' dataset



(d) Average returns on 'Miniboone' dataset



(e) Average cumulative returns on 'Pen Digits' dataset



(f) Average returns on 'Beans' dataset

Fig. 7: Comparison between proposed models performances on validation data

TABLE II: Cumulative Return Performances on test data

	Cumulative Average Returns				
	All Features	RFE + classifier	All Action Updates	SotA	Proposed
Synthetic	$93.68\ \binom{93.8}{93.6}$	$93.68\ \binom{93.72}{93.62}$	$\mathbf{94.52}\ \binom{\mathbf{94.58}}{\mathbf{94.44}}$	$94.16\ \binom{94.26}{94.06}$	$\mathbf{94.50}\ \binom{\mathbf{94.60}}{\mathbf{94.26}}$
Wine	$57.11\ \binom{58.4}{54.8}$	$\mathbf{82.94}\ \binom{\mathbf{85.17}}{\mathbf{80.17}}$	$\mathbf{81.51}\ \binom{\mathbf{83.93}}{\mathbf{78.73}}$	$\mathbf{84.02}\ \binom{\mathbf{88.75}}{\mathbf{78.95}}$	$\mathbf{83.23}\ \binom{\mathbf{88.29}}{\mathbf{77.56}}$
Healthrisk	$24.09\ \binom{26.4}{21.2}$	$\mathbf{55.86}\ \binom{\mathbf{58.25}}{\mathbf{53.04}}$	$\mathbf{57.05}\ \binom{\mathbf{59.26}}{\mathbf{54.87}}$	$\mathbf{57.40}\ \binom{\mathbf{61.20}}{\mathbf{54.00}}$	$47.96\ \binom{52.07}{43.07}$
Miniboone	$-60.55\ \binom{-60.4}{60.8}$	$64.95\ \binom{65.12}{64.77}$	$66.38\ \binom{66.70}{66.05}$	$\mathbf{70.39}\ \binom{\mathbf{72.12}}{\mathbf{68.92}}$	$58.79\ \binom{60.72}{57.30}$
Pen Digits	$48.31\ \binom{48.4}{48.2}$	$61.02\ \binom{61.33}{60.64}$	$66.94\ \binom{67.51}{66.36}$	$\mathbf{73.32}\ \binom{\mathbf{74.17}}{\mathbf{72.67}}$	$69.99\ \binom{70.66}{69.15}$
Beans	$36.85\ \binom{37.3}{36.4}$	$64.63\ \binom{65.07}{64.18}$	$71.12\ \binom{71.56}{70.75}$	$\mathbf{76.19}\ \binom{\mathbf{76.93}}{\mathbf{75.22}}$	$67.19\ \binom{68.10}{66.32}$

Although not outlined, the proposed Transformer-based RL model incurs higher computational overhead during training than other baselines, especially ones without RL. Adopting efficient Transformer variants can substantially reduce training costs [44].

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and the National Council for Scientific and Technological Development – CNPq under project number 314121/2021-8.

REFERENCES

[1] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[2] J. P. Li, A. U. Haq, S. U. Din, J. Khan, A. Khan, and A. Saboor, "Heart disease identification method using machine learning classification in e-healthcare," *IEEE Access*, vol. 8, pp. 107562–107582, 2020.

[3] I. Padhi, Y. Schiff, I. Melnyk, M. Rigotti, Y. Mroueh, P. Dognin, J. Ross, R. Nair, and E. Altman, "Tabular transformers for modeling multivariate time series," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3565–3569, IEEE, 2021.

[4] G. Zaks and G. Katz, "Recom: A deep reinforcement learning approach for semi-supervised tabular data labeling," *Information Sciences*, vol. 589, pp. 321–340, 2022.

[5] M. Kachuee *et al.*, "Cost-sensitive diagnosis and learning leveraging public health data." arXiv preprint, 2019.

[6] M. R. Parvez, T. Bolukbasi, K. W. Chang, and V. Saligrama, "Robust text classifier on test-time budgets." arXiv preprint, 2018.

[7] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[8] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, p. 529–533, 2015.

[9] G. Baruque and W. Caarls, "All action updates for reinforcement learning

TABLE III: Accuracy Performances on test data

	Average Accuracy				
	All Features	RFE + classifier	All Action Updates	SotA	Proposed
Synthetic	$0.998\ \binom{0.999}{0.998}$	$\mathbf{0.998}\ \binom{\mathbf{0.999}}{\mathbf{0.998}}$	$0.997\ \binom{0.998}{0.997}$	$\mathbf{0.999}\ \binom{\mathbf{0.999}}{\mathbf{0.998}}$	$0.998\ \binom{0.998}{0.996}$
Wine	$0.981\ \binom{0.987}{0.969}$	$\mathbf{0.960}\ \binom{\mathbf{0.971}}{\mathbf{0.946}}$	$0.939\ \binom{0.951}{0.922}$	$\mathbf{0.950}\ \binom{\mathbf{0.972}}{\mathbf{0.922}}$	$\mathbf{0.958}\ \binom{\mathbf{0.972}}{\mathbf{0.917}}$
Healthrisk	$0.845\ \binom{0.857}{0.831}$	$0.824\ \binom{0.836}{0.810}$	$\mathbf{0.834}\ \binom{\mathbf{0.845}}{\mathbf{0.823}}$	$\mathbf{0.853}\ \binom{\mathbf{0.873}}{\mathbf{0.831}}$	$0.811\ \binom{0.830}{0.783}$
Miniboone	$0.947\ \binom{0.948}{0.946}$	$0.870\ \binom{0.871}{0.869}$	$0.863\ \binom{0.865}{0.860}$	$0.920\ \binom{0.925}{0.915}$	$\mathbf{0.928}\ \binom{\mathbf{0.929}}{\mathbf{0.926}}$
Pen Digits	$0.967\ \binom{0.967}{0.966}$	$0.940\ \binom{0.942}{0.938}$	$0.922\ \binom{0.926}{0.919}$	$\mathbf{0.956}\ \binom{\mathbf{0.958}}{\mathbf{0.953}}$	$0.951\ \binom{0.955}{0.948}$
Beans	$0.924\ \binom{0.927}{0.922}$	$0.883\ \binom{0.885}{0.881}$	$0.908\ \binom{0.911}{0.906}$	$\mathbf{0.927}\ \binom{\mathbf{0.931}}{\mathbf{0.924}}$	$\mathbf{0.925}\ \binom{\mathbf{0.928}}{\mathbf{0.921}}$

with costly features," in *2023 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pp. 1–6, IEEE, 2023.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[12] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, "Video swin transformer," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 3202–3211, 2022.

[13] Y. Gong, Y.-A. Chung, and J. Glass, "Ast: Audio spectrogram transformer," *arXiv preprint arXiv:2104.01778*, 2021.

[14] G. Dulac-Arnold, L. Denoyer, P. Preux, and P. Gallinari, "Datumwise classification: a sequential approach to sparsity," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, p. 375–390, 2011.

[15] H. Shim, S. J. Hwang, and E. Yang, "Why pay more when you can pay less: A joint learning framework for active feature acquisition and classification." arXiv preprint, 2017.

[16] M. Kachuee, O. Goldstein, K. Karkkainen, S. Darabi, and M. Sarrafzadeh, "Opportunistic learning: Budgeted cost-sensitive learning from data streams." ArXiv preprint, 2019.

[17] J. Janisch, T. Pevný, and V. Lisý, "Classification with costly features using deep reinforcement learning," in *AAAI Conference on Artificial Intelligence*, 2019.

[18] J. Janisch, T. Pevný, and V. Lisý, "Classification with costly features as a sequential decision-making problem," *Machine Learning*, vol. 109, p. 1587–1615, 2020.

[19] F. Nan, J. Wang, and V. Saligrama, "Pruning random forests for prediction on a budget," *Advances in NeurIPS*, vol. 29, 2016.

[20] F. Nan and V. Saligrama, "Adaptive classification for prediction under a budget," *Advances in NeurIPS*, vol. 30, 2017.

[21] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in neural information processing systems*, vol. 34, pp. 15084–15097, 2021.

[22] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.

[23] R. Agarwal, D. Schuurmans, and M. Norouzi, "An optimistic perspective on offline reinforcement learning," in *International conference on machine learning*, pp. 104–114, PMLR, 2020.

[24] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[25] M. Janner, Q. Li, and S. Levine, "Offline reinforcement learning as one big sequence modeling problem," *Advances in neural information processing systems*, vol. 34, pp. 1273–1286, 2021.

[26] Y. Chebotar, Q. Vuong, K. Hausman, F. Xia, Y. Lu, A. Irpan, A. Kumar, T. Yu, A. Herzog, K. Pertsch, *et al.*, "Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions," in *Conference on Robot Learning*, pp. 3909–3928, PMLR, 2023.

[27] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," *arXiv preprint arXiv:2110.06169*, 2021.

[28] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *NIPS Deep Learning Worksho*, 2013.

[30] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[31] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016.

[32] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[33] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[34] K. He, C. Gan, Z. Li, I. Rekik, Z. Yin, W. Ji, Y. Gao, Q. Wang, J. Zhang, and D. Shen, "Transformers in medical image analysis," *Intelligent Medicine*, vol. 3, no. 1, pp. 59–78, 2023.

[35] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI open*, vol. 3, pp. 111–132, 2022.

[36] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.

[37] D. Dua and C. Graff, *UCI Machine Learning Repository (Irvine*. CA: University of California, School of Information and Computer Science, 2019.

[38] A. Singh, "HealthRiskData_Ver_3," 2019.

[39] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor, "Boosted decision trees as an alternative to artificial neural networks for particle identification," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 543, pp. 577–584, may 2005.

[40] F. Alimoglu and E. Alpaydin, "Methods of combining multiple classifiers based on different representations for pen-based handwriting recognition," in *Proceedings of the fifth Turkish artificial intelligence and artificial neural networks symposium (TAINN 96)*, 1996.

[41] M. Koklu and I. A. Ozkan, "Multiclass classification of dry beans using computer vision and machine learning techniques," *Computers and Electronics in Agriculture*, vol. 174, p. 105507, 2020.

[42] E. Parisotto, F. Song, J. Rae, R. Pascanu, C. Gulcehre, S. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, *et al.*, "Stabilizing transformers for reinforcement learning," in *International conference on machine learning*, pp. 7487–7498, PMLR, 2020.

[43] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[44] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Comput. Surv.*, vol. 55, Dec. 2022.