# Integer Sequence Refinement Using Language Models and Reinforcement Learning

1st Carolina Carvalho
Department of Computer Science
University of Évora
Évora, Portugal
carolina.rcxc@gmail.com

2nd Paulo Quaresma
Department of Computer Science
University of Évora
Évora, Portugal
pq@uevora.pt

***Abstract*—This article extends the applicability domain of language models to problems where candidate solutions can be expressed as an encoded integer sequence. Considering this sequence, language models can operate in the neural machine translation setting and leverage their optimization power for heuristic search techniques. Reinforcement Learning (RL) is applied to Language Models (LM), regardless of whether character-level or word-level models are used as a basis. To stabilize the learning, several approaches are explored, including functional and architectural decoupling. The framework is then applied to two combinatorial problems, namely the Traveling Salesman Problem benchmark and Neural Architecture Search, which is used to generate a hierarchical (tree-based) text classifier where the blocks are inspired by the InceptionV1 architecture. The decoupling results are the main contribution of this paper, easing the RL and LM stabilization requirements while expanding the resolution domain beyond Markov Decision Processes to non-causal normative heuristic problems, such as Neural Architecture Search (NAS).**

***Keywords*- *Heuristic Optimization; Reinforcement Learning; Language Model; Task Semantic Segmentation; Artificial Neural Network; Neural Architecture Search; Unordered Markov Decision Processes; Bellman Operator.***

## I. INTRODUCTION

Neural Machine Translation models are capable to generate text by mapping it from the origin language to a target one. Some training metrics which evaluate the translation quality are non differentiable such as the BLEU score, what makes it impossible to use as a training cost function when using Error Back Propagation and bringing this score's usage only to the evaluation stage.

By re-framing performance scores as heuristics, the training process can then be seen as an Heuristic Search methodology. To access this problem is known that the usage of Reinforcement Learning (RL) allows to optimize a non-differentiable metric, the reward, but its restricted to Markov Decision Processes (MDPs). This kind of training, RL, is also known as challenging to train in complex tasks.

This work proposes two decoupling philosophies: architectural and functional as well as solution encoding techniques which can be used in RL training. The proposed decoupling techniques allow to extend the RL training scope to more complex problems which are not MDPs and also enables the usage of more complex models, such as Language Models, in the RL training loop.

Regarding the learning process, it works in a similar to the Generative Adversarial Networks (GANs). By using a specialized critic network which learns the problem's features and rewards a task-preforming network, the agent network such as a professor and a student learning about the same problem.

Considering the sequence encoding of the candidate solutions generated by the language model, an ontology must be defined to encode and serialize the candidates, allowing the architectures to generate data structures and refine them during training, similar to a GAN training setting. In contrast to traditional Heuristic Search methods, where the candidate solution can be an array of various degrees of freedom in the problem (e.g., variables in a multivariate optimization problem), language models can capture the data structure or ontology with the help of special characters. These characters are used in the sequence encoding, signaling the evaluation methodology to build and assess a diversity of data structures. This capability is referred to in this paper as Semantic Encoding. It is then applied to the Neural Architecture Search downstream problem.

The rest of this paper is organized as follows: II. Related Work, III. Proposed Approach, IV. Sequence Encoding, V. Semantic Encoding, VI. Proposed Architectures, VII. Reinforcement Learning as a Search Methodology, VIII. Decoupled Asynchronous Advantage Actor-Critic, IX. Decoupled Soft Q-Learning, X. Decoupling's Mathematical Formalization, XI. Proposed Training Formulation, XII. Accessed Problems, XIII. Results, XIV. Error Analysis. Finally, the paper concludes with XV. Conclusions and Future Work and XVI. Acknowledgments.

## II. RELATED WORK

The present work is an extension of [1], an eKNOW 2025 conference paper with focus on formalizing the approaches.

Regarding the Natural Language Processing domain, the auto-encoder Language Models are typically trained on a large corpus. To evaluate the language model, the pretrained encoder, along with a custom decoder tailored to the downstream task, is then fine-tuned to address the specific task. The encoder part of the language model retains knowledge and maps semantics to a reduced latent dimension. This learned mapping keeps, in the encoder's weights, general type features, such as how to speak a language. This work explores the language model's encoder capability to retain the semantics of other problems beyond merely speaking a language. Additionally, the generative capability of language models is examined.

There are instances where the intention is to model the dataset's probability density function rather than the data itself. For example, in generative models, the goal is to generate data similar to the dataset. To achieve this objective, variational models come into play, specifically Variational Auto-Encoders (VAE) [2]–[12] and Generative Adversarial Network (GAN) architectures [13]–[19]. The GAN architectures use a Generator and a Discriminator network and employ min-max training. During training, the generator network produces data samples of better quality at each time step to trick the discriminator, which learns to distinguish real data from fake data generated by the Generator network. In this manner, both networks engaged in min-max training learn to perform their respective tasks. The Generator produces more realistic data samples as the Discriminator becomes increasingly difficult to deceive. In terms of VAEs, these models approximate the dataset's probability density function by modeling its parameters [20] or by assigning an odds to each output [21], generating data from the random variable where each output holds the model's estimated odds. The resulting binary text classifier positioning of a dataset, this work posits that any problem for which the solutions can be encoded in an integer sequence can also be addressed in a generative manner. It is essential to assert that the optimization goal is expressible through a heuristic function, akin to the fitness function in the context of Genetic Algorithms (GAs) [22]–[31]. Heuristic search using Language Models suffers from a lack of exploration due to the well-known difficulty of stabilizing complex neural models when trained using Reinforcement Learning. Traditional issues include training convergence and subsequent hyperparameter tuning. Furthermore, RL is usually applied sequentially to causal problems. This paper proposes decoupling-based RL training techniques and network architecture design principles that enable the application of RL to new problem types, as well as the incorporation of Language Models' feature capture capabilities to address problems beyond linguistic ones.

The proposed heuristic search relates mainly to evolutionary algorithms, such as Genetic Algorithms. The adopted models are neural Language Models, and the training is based on Reinforcement Learning. In this section, all the aforementioned methods are detailed. Evolutionary algorithms can be seen as heuristic search engines in the sense that they generate candidate solutions, which are evaluated on the fly using a heuristic function, such as the fitness function in the case of Genetic Algorithms. Neural Language Models (LMs) are used for language modeling [32]–[39]. They learn meaningful features from text data through embedding generation techniques. When an LM is used in the context of Neural Machine Translation [37], [40]–[45], LMs can be viewed as generative models because they generate tokens that, when decoded, are words in the target language domain. This problem can be generalized into a Sequence2Sequence problem when considering the same language model architecture generating a sequence with a different semantic encoding than target language tokens, always restricted to a differentiable loss function. Neural Machine Translation (NMT) architectures are generative by nature because they produce tokens in the model's target language, although their training typically requires a differentiable loss function that might not accurately express the training goals. The same occurs in Neural Architecture Search (NAS) tasks, where the primary objective is to increasingly enhance the candidate network's performance metric. In [46], a Recurrent Neural Network (RNN) is trained using Reinforcement Learning (RL) with the candidate network's performance almost directly serving as the reward function, employing various techniques to reduce the training's variance and facilitate learning through the described method. To relax the differentiable metric constraint, a new type of training is necessary; this is where Reinforcement Learning (RL) becomes relevant. RL techniques are primarily based on Markov Decision Processes (MDPs). Several training approaches attempt to optimize non-differentiable metrics in a deep model, such as surrogate losses [47], minimum risk training [48], and reinforcement learning [46]. All these training methodologies have their limitations: surrogate losses and reinforcement learning are difficult to stabilize, and minimum risk training is too computationally expensive when applied to a language model like an NMT architecture. Focusing on RL training, this article explores methods to stabilize the training and establish a robust optimization framework.

## III. Proposed Approach

To ease the training, several models are used inside the training loop. Both character-level and word-level language models are explored with different architectural and functional decoupling strategies. Regarding the character-level language model, it is proposed in [46] and is based on one-dimensional convolutional layers. In the proposed architecture, two models are used in this Soft Q-Learning training loop: one for acting on the environment (the target network) and another that learns the Soft Q-values. The models have the same architecture, so the weights with the learned Q-values can be set in the target network. In this way, transfer learning occurs during training. The problem of heuristic sequence building, also known as refinement, is broken into position and value generation; therefore, each model has two dedicated outputs. One output generates the position of the new element in the sequence, while the other generates the assigned value. In this way, there are dedicated architectural components: the core network learns the problem features and has connected dedicated Dense layers for each position and value generation, respectively. Notice that since Soft Q-Learning is being used, the target network outputs are odds, so a random experiment with the calculated odds distribution should be performed. Also, the SoftQ-Net has the same core architecture and learns the Q-values for the position and value parameters using the same core architecture as the Char-Conv. Thus, the target network has two learning sources: from interaction with the environment and from the transferred weights of the Q-Network.

For the word-level language model, the Transformer's encoder [49] was chosen to integrate into the Vector Quantized Variational Autoencoder proposed architecture. The quantized layer is derived from [49], and the dedicated outputs are based on Dense layers. In this case, one full model generates the position, while another is used for the value. Each full model is a VQ-VAE with two dedicated outputs: one for the actor and another for the critic's output. The two complete models are trained with a dedicated gradient tracker, but as they belong to the same problem, the reward and the state are the same for both models. One tracking the position and the other the value. The training of each VQ-VAE follows the Asynchronous Advantage Actor-Critic (A3C) methodology. By using the Vector Quantized layer proposed in [50], the search space is divided while the Transformer's encoder learns the problem's features in a parallelized manner, reducing the amount of time needed to explore it. An epsilon-greedy technique is also used to boost the algorithm's exploration. In this case, the model has a dedicated architectural core, the encoder, to learn the problem features, which are shared between the actor and the critic outputs. Consequently, the model learns from the actor-critic dynamic and through the heuristic reward. It is important to note that, in this case, the actor's outputs represent odds, and with this probabilistic representation, a random experiment is performed to choose the final selected action. The next step is to explain why does this decoupled architecture learn using RL.

### A. Problem Formulation

Considering a Markov Decision Process (MDP) with an initially unknown functioning that is learned using Reinforcement Learning and Artificial Neural Networks. This decision process can be of difficult representation because of it's complexity and this uncertainty can be modeled with randomness for more complex problems. Usually the state space is large and therefor its beneficial to use compression techniques such as latent spaces leanings as well as to make probabilistic learning by considering the model output as odds in a random experiment.

Usually in RL, the learning process is made incrementally and sequentially, what forces causality in the ANN model which is not desirable in some downstream problems such as Neural Architecture Search (NAS), where this incremental and sequential behavior does not adequately the problem dynamics. In such cases, decoupling the state generation task into position and value generation in order to change the next value in the sequence. This decoupling methodology allows to extend the RL training's applicability to larger problems which are not necessarily causal.

Next the impact of position-value decoupling, which is formally known as factorization, is accessed.

*1) Classical MDPs and Temporal Dependency:* Starting from classical MDPs:

$$M = (S, A, P, R, \gamma)$$

Being $(S, A, P, R, \gamma)$ the state, action, policy and reward spaces and $\gamma$ the learning factor. A policy $\pi$ maps states to actions, and the goal is to find $\pi^*$ maximizing:

$$J(\pi) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right]$$

The value function $V^\pi(s)$ and action-value function $Q^\pi(s, a)$ satisfy:

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[Q^\pi(s, a)]$$
$$Q^\pi(s, a) = R(s, a) + \gamma\mathbb{E}_{s'\sim P(\cdot|s,a)}[V^\pi(s')]$$

This framework is inherently sequential: each decision depends on the prior state and action.

*2) Decomposed MDP via Value-Position Factorization:* Now we introduce a decomposed representation of an MDP trajectory

$$\tau = \{(s_t, a_t, r_t)\}_{t=0}^{T}$$

into position-independent events.

**Definition (Decomposed MDP):**

Define the set:

$$Z_\tau = \{Z_t = (s_t, a_t, Q_t) \mid t = 0, \ldots, T\}$$

where:

$$Q_t := Q^\pi(s_t, a_t)$$

Time $t$ is now a latent index, not an explicit generator.

We view the trajectory as a set:

$$Z_\tau \in \mathcal{P}(S \times A \times \mathbb{R})$$

This is now a permutation-invariant object; that is, it lies in the space of unordered tuples, and hence non-sequential modeling is possible.

*3) RL Objective on Decomposed MDPs:*

*Step 1: Define the Learning Objective:* We define a modified objective:

$$J_{\text{decomp}}(\pi) = \mathbb{E}_{Z_\tau\sim\pi} \left[ \sum_{(s,a,Q)\in Z_\tau} w(s, a)Q \right]$$

where $w(s, a)$ is a weighting function (possibly uniform or importance-weighted).

This allows learning over unordered samples, with Q-values acting as surrogate return signals.

*Step 2: Bellman Consistency Holds:* We now prove: Bellman consistency can be enforced over unordered samples.

Let $Z_t = (s_t, a_t, Q_t)$, and suppose we estimate:

$$Q_\theta(s, a) \approx R(s, a) + \gamma\mathbb{E}_{s'\sim P(\cdot|s,a)}[V_\theta(s')]$$

Define the loss:

$$L(\theta) = \mathbb{E}_{(s,a,Q)\sim\mathcal{D}} \left[ (Q_\theta(s, a) - Q)^2 \right]$$

If the samples $(s, a, Q)$ are drawn from any replay buffer or dataset, regardless of temporal order, the loss is still valid because the Bellman operator:

$$\mathcal{T}Q(s, a) := R(s, a) + \gamma\mathbb{E}_{s'} \left[ \max_{a'} Q(s', a') \right]$$

is defined pointwise.

This justifies learning Q-values independently of temporal order, so long as:

- $Q(s, a)$ is sampled accurately
- The agent uses valid updates

Now the above mentioned usage of odd regression in the neural architectures' output while making non-sequential sequence generation is analyzed.

*4) Enabling Odd Regression (Non-Sequential Generation):* Let's define *odd regression* as the ability to predict or generate parts of a trajectory non-sequentially, e.g., generating action $a_t$ before observing $a_{t-1}$.

NON-SEQUENTIAL LEARNING IN VALUE-DECOMPOSED MDPS

*Definitions*

Let an MDP be defined as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ with:

- $s_t \in \mathcal{S}, a_t \in \mathcal{A}$
- $P(s_{t+1} \mid s_t, a_t)$: transition kernel
- $r_t = R(s_t, a_t)$
- $\pi(a \mid s)$: stationary policy
- $Q^\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R(s_k, a_k) \,\middle|\, s_0 = s, a_0 = a \right]$

Define a trajectory $\tau = \{(s_t, a_t, r_t)\}_{t=0}^T$ and its value-decomposed representation:

$$Z_\tau = \{Z_t := (s_t, a_t, Q_t) \mid Q_t = Q^\pi(s_t, a_t)\}_{t=0}^T$$

as an *unordered multiset*. We allow access to any subset $Z_{\setminus t}$ for conditional generation or regression.

*Step 1: Bellman Operator is Pointwise*

The Bellman operator $T^\pi$ is pointwise:

$$\begin{aligned} Q^\pi(s, a) &= (T^\pi Q^\pi)(s, a) \\ &:= R(s, a) + \gamma \, \mathbb{E}_{s' \sim P(\cdot|s,a)} \mathbb{E}_{a' \sim \pi(\cdot|s')} \left[ Q^\pi(s', a') \right] \end{aligned} \tag{1}$$

By definition, $(T^\pi Q)(s, a)$ depends only on the local state-action pair $(s, a)$ and the expected future values. That is, for any trajectory $\tau$, the update at $(s, a)$ does not require sequential access to previous or future transitions:

$$Q^\pi(s, a) = R(s, a) + \gamma \, \mathbb{E}_{s', a'} \left[ Q^\pi(s', a') \right] \tag{2}$$

Thus, learning $Q^\pi$ via Temporal DIfference (TD) methods or fitted Q-iteration is inherently *pointwise* and does not require temporal ordering. □

*Step 2: Learning from Unordered Sets*

Consider the Q-learning loss over a dataset $\mathcal{D}$ of transitions sampled in arbitrary order:

$$\begin{aligned} \mathcal{L}_Q(\theta) := \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \Big[ \big( & Q_\theta(s, a) - r \\ & - \gamma \, \mathbb{E}_{a' \sim \pi(\cdot|s')} Q_\theta(s', a') \big)^2 \Big] \end{aligned} \tag{3}$$

Minimizing $\mathcal{L}_Q$ over unordered tuples $(s, a, r, s')$ yields convergence to $Q^\pi$ under standard assumptions (ergodicity, coverage, Robbins-Monro conditions).

This follows directly from the convergence of TD learning and fitted Q-iteration. The temporal index $t$ is irrelevant: each update applies the Bellman operator $T^\pi$ pointwise. Therefore, transitions may be sampled in any order without affecting convergence. □

*Step 3: Permutation-Invariant Conditional Decoding*

Let a decoder $\pi(Z_t \mid Z_{\setminus t})$ be trained via the set-based loss:

$$\mathcal{L}_{\text{set}} := \mathbb{E}_{Z_\tau} \sum_{Z_t \in Z_\tau} - \log \pi(Z_t \mid Z_{\setminus t})$$

[Non-Sequential Generation is Bellman-Consistent] Let $Z_\tau = \{(s_t, a_t, Q^\pi(s_t, a_t))\}_{t=0}^T$. If

1) $Q_\theta$ is learned with Bellman-consistent TD updates,
2) $\pi(Z_t \mid Z_{\setminus t})$ is trained to predict individual transitions from context,

then

- The decoder can generate valid trajectory components in arbitrary order,
- Q-values remain consistent with the Bellman equations,
- Reinforcement learning objectives are correctly optimized even without sequential ordering.

Each $Z_t = (s_t, a_t, Q_t)$ satisfies $Q_t = Q^\pi(s_t, a_t)$ by pointwise TD learning. The conditional decoder $\pi(Z_t \mid Z_{\setminus t})$ is permutation-invariant: sampling any $Z_t$ conditioned on the rest respects the learned distribution of state-action-value triples.

Because $Q$-values are Bellman-consistent:

$$Q_t = R(s_t, a_t) + \gamma \mathbb{E}_{s', a'} [Q_\theta(s', a')]$$

any sampled element from $\pi$ will satisfy the Bellman equation relative to other elements. Therefore, one can reconstruct valid trajectories non-sequentially or fill missing entries conditionally. □

## IV. SEQUENCE ENCODING

The sequence which, in both of the proposed approaches, is the RL state and is refined during the training process, can have a learnable structure or ontology. For example, it can be a serialized image or text or an optimization problem candidate solution. For example, if a solution with a maximum accepted length is required, a padding char can be used. Special chars can be used to guide the solution's builder and evaluator and generate a proper reward according to it's performance, similarly to what is made in other Heuristic Search methodologies such as Genetic Algorithms.

In this document two problems were accessed, Neural Architecture Search (NAS) where the sequence is the Network Structure Code (NSC) and the reward is the built classifier's accuracy. The second problem is the Traveling Salesman Problem (TSP) in which the sequence is an ordered succession of visited cities. The RL reward is then an inverted total distance.

## V. Semantic Encoding

Sequence semantic encoding is one of the core subjects in this proposal. When applied to the sequence generated by a Neural Machine Translation model, the problem can be transposed into an optimization problem where the candidates can be encoded as a sequence of integers [46]. The candidate solutions' meta-format can be a single value or a sequence of values, depending on the downstream problem. Special characters such as separators or sequence terminators can also be used to help specify the solution's evaluator behavior. The optimization problem structure that this kind of semantic encoding enables is a heuristic search, since the candidate solution's quality is evaluated by a reward function that can be non-differentiable, and its value can be generated during the search execution.

For example, in order to access the Neural Architecture Search (NAS) problem using the proposed technique, the sequence can be the Network Structure Code (NSC), which encodes the candidate neural network hyperparameters. The network is then built and trained so that the performance metric can be extracted and the candidate sequence evaluated. Figure 1 highlights the proposed heuristic search architecture.

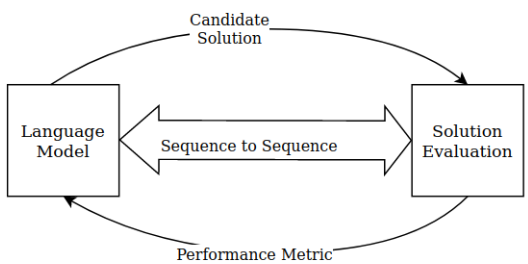



Figure 1: Heuristic Search Architecture: Similar to an evolutionary algorithm, this neural network–based candidate-solution refinement technique allows the solution evaluation block to be non-differentiable. This block only needs to produce an empirical performance metric, such as a RL reward.

## VI. Proposed Architectures

Depending on the nature of the problem, it can be beneficial to generate the sequence iteratively or through composition. As this article's subject is the usage of language models in optimization problems, and language models can encode semantics based on characters or words, both approaches will be explored further.

With the RL training enabled by the decoupling, based on unitary and semantically segmented tasks assigned to unitary model parts, the proposed architectures consist of two models inspired by character-level and word-level language models, respectively. With this training possibility, these models' generalization capability, as well as the proposed modeling principle, will be assessed.

### A. Transformer-based Vector Quantized Variational Auto-Encoder with Asynchronous Advantage Actor Critic

The word-level model is based on the Transformer architecture proposed in [49], which includes both the Transformer encoder and decoder architectures, along with the vector quantization layer proposed in [21]. This Vector Quantized Variational Autoencoder (VQ-VAE) architecture was decoupled as well; however, in this case, its outputs correspond to the actor and the critic. The actor outputs log probabilities for the possible actions of the RL agent, and the critic rates the inputs. Maintaining the sequence composition decoupling strategy, two models are employed to compose the target sequence. Once again, one model specializes in generating the position, while the second model generates the value to be assigned. Each model has two dedicated outputs, one to act an another for the critic while both share the core encoder that captures features from data. Figure 2 illustrates the architecture utilized for the VQ-VAE.

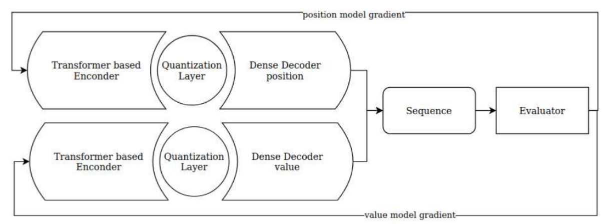



Figure 2: Decoupled Vector Quantized Variational Auto-Encoder proposed architecture using A3C. The encoder comes from the Transformer architecture, the quantization layer from [21], and the decoder is made of stacked dense layers. In this training architecture, each VQ-VAE has a actor output and a critic output. Two VQ-VAEs are represented in this figure and each one learns how to generate the value and position of the new element in the sequence.

### B. Char-Conv with DeepQNet-Policy Learning

Starting from the model proposed in [51], two output kernels were used to decouple the tasks into position and value generation. In this way, one model pair, Q-Network and Policy-Net, is used to compose the candidate sequence. Regarding the Traveling Salesman Problem, a benchmark problem, the proposed training setting works without issues. When considering the Neural Architecture Search problem, the reward signal presents high variance and the training did not converge to zero. In addressing this problem, two changes were made: entropy regularization was added, and the output activation function was changed to linear so that the model output is interpreted as log probabilities for each output position.

## VII. Reinforcement Learning as a Search Methodology

Since the search for optimal solutions is guided by reinforcement learning, the model generates multiple candidate solutions and iteratively improves them based on feedback. A heuristic evaluation function $g : \mathcal{Y} \to \mathbb{R}$ assigns a quality score to candidate solutions, serving as a reward signal:

$$R(y) = g(y). \tag{4}$$

Given any problem where the solution space $\mathcal{Y}$ is structured as integer sequences, the proposed methodology guarantees:

- **Expressibility**: The model $\hat{f}_\theta$ can learn to generate valid sequences from $\mathcal{X}$ using neural networks that are trained on $\mathcal{D}$.
- **Optimization Capability**: The reinforcement learning-based search ensures that generated solutions are iteratively improved using $g(y)$.
- **Generalization**: The auto-regressive nature of the model allows it to generate variable-length solutions applicable to different instances of the problem since a special character can be used as a sequence terminator.

Thus, for any integer-encoded problem, the formulation is sufficient to obtain high-quality solutions through iterative refinement. The proposed formulation applies to a wide range of problems where solutions are represented as integer sequences, including:

- Combinatorial optimization problems (e.g., the Traveling Salesman Problem, Knapsack Problem).
- Scheduling and planning tasks where actions are encoded as integer sequences.
- Code synthesis and symbolic regression.
- Game strategies with discrete action spaces.
- Non-sequential problems that benefit from value-position decoupling.

For any such problem, the integer-encoded representation ensures that the model can map problem instances to structured sequences and refine them over iterations using reinforcement learning. The search methodology follows a reinforcement learning-based approach such as DQN-PL [52], [53], A3C [54], and SoftQ-Learning [55]. The exploration methodology is epsilon-greedy for all the approaches. The different training methodologies are described in the next subsections.

## VIII. Decoupled Asynchronous Advantage Actor Critic

The main concept in decoupling is to create a problem feature extraction core and decoupled output decoders to model the output value according to the problem's required output. For example, in the VQ-VAE with the A3C training case, the same model generates the action and its corresponding critic value. To generate a sequence, two models with the specified decoupling are used: one generates the position of the new element, and the second generates its value. The resulting sequence is then updated and iteratively refined. Next, the formal formulation of this kind of decoupling is provided.

### A. Policy and Value Functions

Let $S$ be the state space, $A$ be the action space, and $P(s'|s,a)$ be the transition probability. The reward function is defined as $R(s,a,p)$, where $p$ is the selected position.

The policy consists of two independent components:

$$\pi(a|s;\theta_a) \quad \text{and} \quad \pi(p|s;\theta_p) \tag{5}$$

where:

- $\pi(a|s;\theta_a)$ selects an action based on state $s$.
- $\pi(p|s;\theta_p)$ selects a position based on state $s$.

The value functions are defined as:

$$V_{\text{act}}(s;\theta_v) = \mathbb{E}\left[R(s,a,p) + \gamma V_{\text{act}}(s')\right] \tag{6}$$

$$V_{\text{pos}}(s;\theta_p) = \mathbb{E}\left[R(s,a,p) + \gamma V_{\text{pos}}(s')\right] \tag{7}$$

### B. Exploration-Exploitation Strategy

The exploration rate for both action and position selection follows an epsilon-greedy decay:

$$\epsilon_a(t+1) = \max(\epsilon_a(t) \cdot d, \epsilon_{\min}) \tag{8}$$

$$\epsilon_p(t) = \epsilon_a(t) \tag{9}$$

where $d$ is the decay factor and $\epsilon_{\min}$ is the minimum exploration rate.

### C. Advantage Function and Returns

The advantage function for actions is given by:

$$A_{\text{act}}(s,a) = r + \gamma V_{\text{act}}(s') - V_{\text{act}}(s) \tag{10}$$

The advantage function for positions is:

$$A_{\text{pos}}(s,p) = r + \gamma V_{\text{pos}}(s') - V_{\text{pos}}(s) \tag{11}$$

The discounted return at timestep $t$ is:

$$G_t = \sum_{k=0}^{T-t} \gamma^k R(s_{t+k}, a_{t+k}, p_{t+k}) \tag{12}$$

The returns are then normalized:

$$\hat{G}_t = \frac{G_t - \mu(G)}{\sigma(G) + \epsilon} \tag{13}$$

### D. Loss Functions

The critic losses for action and position value networks are:

$$L_{\text{critic-act}} = \sum_t \left(A_{\text{act}}(s_t, a_t)\right)^2 \tag{14}$$

$$L_{\text{critic-pos}} = \sum_t \left(A_{\text{pos}}(s_t, p_t)\right)^2 \tag{15}$$

The actor losses are:

$$L_{\text{actor-act}} = -\sum_t \log \pi(a_t|s_t) A_{\text{act}}(s_t, a_t) \tag{16}$$

$$L_{\text{actor-pos}} = -\sum_t \log \pi(p_t|s_t) A_{\text{pos}}(s_t, p_t) \tag{17}$$

The total losses are:

$$L_{\text{total-act}} = L_{\text{actor-act}} + L_{\text{critic-act}} \tag{18}$$

$$L_{\text{total-pos}} = L_{\text{actor-pos}} + L_{\text{critic-pos}} \tag{19}$$

### E. Gradient Updates

Gradients for action and position networks are computed separately:

$$\nabla_{\theta_a} L_{\text{total-act}} = \sum_t \nabla_{\theta_a} L_{\text{total-act}} \tag{20}$$

$$\nabla_{\theta_p} L_{\text{total-pos}} = \sum_t \nabla_{\theta_p} L_{\text{total-pos}} \tag{21}$$

These gradients are applied using an optimizer:

$$\theta_a \leftarrow \theta_a - \alpha \nabla_{\theta_a} L_{\text{total-act}} \tag{22}$$

$$\theta_p \leftarrow \theta_p - \alpha \nabla_{\theta_p} L_{\text{total-pos}} \tag{23}$$

where $\alpha$ is the learning rate.

This content was generated with the help of generative artificial intelligence.

## IX. Decoupled SoftQ-Learning

Regarding the CharConv model in the NAS problem assessment, it was not possible to stabilize the training using the traditional DQN-PL approach. In the NAS setting, it was found beneficial for training stability to use stochastic outputs followed by a random experiment with the model's predicted output odds to generate the predicted action. To help stabilize the training in a stochastic environment, entropy regularization was employed.

Concerning the decoupling technique used in this context, two models were utilized. One model features a CharConv core and two decoupled outputs: one for value and another for the position of the new element in sequence generation. The second model is the target network, which generates the stochastic SoftQ-values for each output.

Additionally, an epsilon-greedy exploration strategy was applied in conjunction with an experience replay buffer. The proposed SoftQ-Learning approach uses a different decoupling when compared to the method presented in the previous section. This is specified in the subsequent subsections.

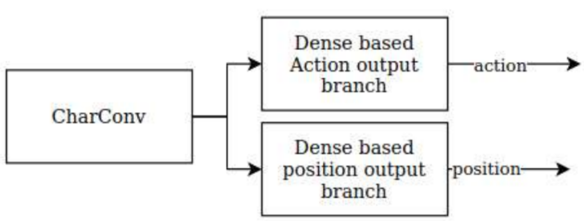



Figure 3: Proposed Decoupling SoftQ-Learning using character-based language model. This architecture is used twice in the training loop, one is specialized in generating new values for the action and position in the sequence to refine, the target-network and the other generates their corresponding Q-values, the Q-network.

### A. State and Action Representation

Let $s \in \mathcal{S}$ be the state space and $a \in \mathcal{A}$ be the action space. Additionally, let $p \in \mathcal{P}$ denote the position selection space. The agent selects both an action and a position at each time step.

### B. Soft Q-Function

Define the Q-function as:

$$Q(s, a, p) = Q_{\text{action}}(s, a) + Q_{\text{position}}(s, p). \tag{24}$$

This decoupling allows independent learning of action and position values.

### C. Soft Q-Learning Update Rule

The update rule follows the soft Bellman equation:

$$\begin{aligned} Q_{\text{action}}(s, a) &\leftarrow (1-\alpha) Q_{\text{action}}(s, a) \\ &+ \alpha \left[ r + \gamma\tau \log \sum_{a'} \exp\left( \frac{Q_{\text{action}}(s', a')}{\tau} \right) \right], \end{aligned} \tag{25}$$

$$\begin{aligned} Q_{\text{position}}(s, p) &\leftarrow (1-\alpha) Q_{\text{position}}(s, p) \\ &+ \alpha \left[ r + \gamma\tau \log \sum_{p'} \exp\left( \frac{Q_{\text{position}}(s', p')}{\tau} \right) \right]. \end{aligned} \tag{26}$$

where:

- $\alpha$ is the learning rate,
- $\gamma$ is the discount factor,
- $\tau$ is the temperature parameter for soft Q-learning,
- $r$ is the received reward,
- $s'$ is the next state.

### D. Action and Position Selection

The action and position are selected using the softmax policy:

$$P(a|s) = \frac{\exp(Q_{\text{action}}(s, a)/\tau)}{\sum_{a'} \exp(Q_{\text{action}}(s, a')/\tau)}, \tag{27}$$

$$P(p|s) = \frac{\exp(Q_{\text{position}}(s, p)/\tau)}{\sum_{p'} \exp(Q_{\text{position}}(s, p')/\tau)}. \tag{28}$$

This formulation allows efficient and structured learning by decoupling position and value, improving performance in reinforcement learning tasks that require both action selection and spatial positioning.

In the next section the position-value decoupling for integer-based sequences is formalized.

## X. Decoupling's Mathematical Formalization

When considering an iteratively generated sequence, in which the elements are generated one after another, the position is fixed and incremental, which implies causality in the sequence generation. By decoupling the functionality into position generation and value generation, thereby composing a single sequence (RL state), it is possible to break the causality implication and still utilize the reinforcement learning capability of optimizing heuristic functions. In this article, the decoupling is achieved at an architectural level; in a multi-branch architecture, each output branch is responsible for one single decoupled task in the non-causal sequence generation. To optimize a single sequence using two models, the state must be shared, and the RL techniques must still be applied to each model, utilizing separate optimizers guided by the same resulting reward.

In incremental sequence generation, this type of sequence generation allows for imposing causality in the RL agent's behavior, leading to a succession of actions generated throughout the training. Regarding compositional sequence generation, where the problem focus is to generate a candidate answer encoded in the sequence rather than a set of actions, decoupling can come into play to divide and conquer the generation problem into two sub-problems, enabling the composition of the sequence without needing to condition on the previous actions.

To extend RL beyond causal MDPs, we decompose the Q-function as follows:

$$Q(s,a) = P(s) + A(s,a), \tag{29}$$

where:

- $P(s) = \mathbb{E}[R|s]$ is the **position value**, which captures the expected reward at state $s$ independent of actions.
- $A(s,a) = Q(s,a) - P(s)$ is the **advantage function**, representing the additional benefit of taking action $a$ beyond merely being in state $s$.

If actions have no influence (a fully non-causal setting), then $A(s,a) = 0$, reducing RL to pure statistical inference:

$$V(s) = P(s) = \mathbb{E}[R|s]. \tag{30}$$

The objective function is defined as:

$$J(\pi) = \mathbb{E}_{s\sim D}[P(s)], \tag{31}$$

where $D$ is a dataset of observed states and rewards. If actions have partial influence, it is optimized as follows:

$$J(\pi) = \mathbb{E}_{s,a\sim D}[P(s) + A(s,a)]. \tag{32}$$

This formulation bridges RL and supervised learning, enabling RL in non-causal settings, such as:

- Counterfactual reasoning.
- Offline and batch RL.
- Decision-making in complex, non-Markovian environments.

## XI. Proposed Training Formulation

In this section, two training algorithms for Heuristic Optimization are proposed: the VQ-VAE model with A3C training and Char-Conv with DQNet-PL, so both character-level and word-level language models are explored.

We define the problem as a Markov Decision Process (MDP) with:

- State space: $S$
- Action space: $A$
- Transition dynamics: $P$
- Reward function: $R$

The objective is to learn a policy $\pi$ that maximizes the cumulative expected reward.

### A. State Representation

The state at time $t$, denoted as $s_t$, represents the environment state:

$$s_t \in S. \tag{33}$$

### B. Action Selection

A neural network models the probability distribution for action selection:

$$a_t \sim \pi(a_t|s_t;\theta). \tag{34}$$

The chosen action $a_t$ is sampled from this distribution.

### C. Critic Network (Value Estimation)

A critic network estimates the value function $V(s_t)$, representing the expected return from state $s_t$:

$$V(s_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right]. \tag{35}$$

### D. Reward and Return Calculation

The immediate reward $r_t$ is received from the environment. The discounted return is computed as:

$$G_t = r_t + \gamma G_{t+1}. \tag{36}$$

The returns are then normalized:

$$\hat{G}_t = \frac{G_t - \mu}{\sigma + \epsilon}. \tag{37}$$

### E. Advantage Estimation

The advantage function measures how much better the taken action was compared to the expected return:

$$A_t = \hat{G}_t - V(s_t). \tag{38}$$

### F. Actor-Critic Loss Functions

The loss for the actor (policy gradient) is:

$$L_{\text{actor}} = -\sum_t \log \pi(a_t|s_t) A_t. \tag{39}$$

The critic is updated using the Huber loss:

$$L_{\text{critic}} = \sum_t \text{Huber}(V(s_t), \hat{G}_t). \tag{40}$$

The Huber loss is defined as:

$$\text{Huber}(x,y) = \begin{cases} \frac{1}{2}(x-y)^2, & \text{if } |x-y| < \delta \\ \delta(|x-y| - \frac{1}{2}\delta), & \text{otherwise} \end{cases} \tag{41}$$

### G. Gradient Update

The gradients of the total loss function are computed as:

$$\nabla_\theta L_{\text{total}} = \nabla_\theta (L_{\text{actor}} + L_{\text{critic}}). \tag{42}$$

The parameters are updated using the Adam optimizer:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L_{\text{total}}. \tag{43}$$

### H. Termination Criteria

Training stops when the running reward exceeds a threshold:

$$\sum_t r_t > R_{\text{target}}. \tag{44}$$

This indicates that the agent has effectively learned an optimal policy for the given task. In this A3C setting, two models are used in order to generate the sequence. In each step a new value and its corresponding position in the sequence (RL state) are generated. Each model has two outputs: one for the action and another for the critic score.

### I. Char Conv + DQN-PL

#### 1. Q-Function Approximation

We approximate the action-value function (Q-function) by a neural network with parameters $\theta$:

$$Q(s, a; \theta) \approx \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s, a_0 = a\right],$$

where:

- $s$ is the state,
- $a$ is the action,
- $r_t$ is the reward at time $t$,
- $\gamma$ is the discount factor.

#### 2. Experience Replay

Experiences are stored in a replay buffer as tuples:

$$(s, a, r, s', d),$$

where $d$ is an indicator that equals 1 if $s'$ is terminal and 0 otherwise.

A mini-batch of $N$ experiences is sampled uniformly at random from the replay buffer for training.

#### 3. Target Calculation

For each sampled experience $(s, a, r, s', d)$, the target value $y$ is computed as:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-) \cdot (1 - d),$$

where $\theta^-$ denotes the parameters of the target network, which are periodically updated to match the primary network parameters $\theta$.

#### 4. Loss Function

The loss function for a mini-batch is defined as the mean squared error between the target and the current Q-value estimate:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - Q(s_i, a_i; \theta)\right)^2.$$

This loss is minimized to update the parameters $\theta$ of the Q-network.

#### 5. Gradient Descent Update

The parameters $\theta$ are updated via gradient descent:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta),$$

where $\alpha$ is the learning rate.

#### 6. Action Selection (Epsilon-Greedy Policy)

At each step, the action $a$ is chosen according to the epsilon-greedy strategy:

$$a = \begin{cases} \text{random action}, & \text{with probability } \epsilon, \\ \arg\max_{a'} Q(s, a'; \theta), & \text{with probability } 1 - \epsilon, \end{cases}$$

with $\epsilon$ decaying over episodes from an initial value $\epsilon_{\text{start}}$ to a minimum value $\epsilon_{\text{min}}$.

#### 7. Periodic Target Network Update

Every fixed number of episodes (or steps), the target network parameters are updated by copying the weights from the primary network:

$$\theta^- \leftarrow \theta$$

## XII. Accessed Problems

For each of the two described ways to generate sequences, causal or non-causal, and regarding the Reinforcement Learning (RL) usage along with the proposed architectures, one benchmark problem was selected. The Traveling Salesman Problem (TSP) for causal generation and Neural Architecture Search (NAS) for non-causal generation.

### A. Traveling Salesman Problem

The TSP consists of generating a tour from a given starting city that passes through all the other cities while minimizing the overall path distance. The considered cities have the following coordinates:

TABLE I: Cities' coordinates used in the Traveling Salesman Problem.

| X | Y |
|---|---|
| 23 | 45 |
| 57 | 12 |
| 38 | 78 |
| 92 | 34 |
| 45 | 67 |
| 18 | 90 |
| 72 | 55 |
| 66 | 24 |
| 83 | 62 |
| 49 | 40 |

A distance matrix is calculated based on the euclidean distance between all the cities. A boolean array is used to track the cities already visited. If a generated city is already visited, the reward function gets the value of -100, in contrary, if a city is not visited, then the reward function gets the value given by:

$$\text{normalized\_reward} = 100 \cdot \left(1 - \frac{\text{distance}}{\text{max\_distance}}\right)$$

With:

$$\text{distance} = \text{distance\_matrix}[\text{current\_city}][\text{action}]$$

The cities road is generated iteratively, one city after another until the generated city is already visited. When this final condition is met, the obtained road is evaluated and the current episode ends.

### *B. Neural Architecture Search*

For the NAS problem, the sequence is interpreted as the Network Structure Code (NSC), meaning that it encodes an Artificial Neural Network (ANN). In this case, it is intended to generate a neural text classifier architecture built by several InceptionV1 blocks [56]. The NSC is composed by two decoupled models which contribute to the same RL final state, also known as NSC. The reward function is the child-network training accuracy which, in the current problem's case, is a classifier network. This classifier is built from an inverted n-ary tree encoded in Depth-First-Search (DFS).

## XIII. Results

In this section, the performance plots for the NAS problem are presented. The adopted search space is an encoded n-ary tree using Depth First Search (DFS). The tree is encoded using $[0, 1, 2]$ in a sequence with a maximum of five positions. A zero encodes a change in the tree branch, a one encodes a deeper instruction, and the two is interpreted as a padding character. Each tree element is a Conv1D version of an InceptionV1 block [56]. When constructed, the tree is inverted so that the root node represents the classifier's final decision kernel. The search focuses on a text classifier, where the embeddings are provided by a Keras embedding layer. For evaluation purposes, this layer is replaced by the RoBERTa large model from Hugging Face [57], achieving state-of-the-art results with the IMDB sentiment analysis dataset [58]. The resulting model from the search was trained using a learning rate scheduler and presents the training curves shown in Figure 3.


Figure 4: Classification accuracy and binary cross-entropy loss when using the generated NAS classifier and RoBERTa as embedding model.

The resulting binary text classifier positioning in the state of the art is presented in Table III.

TABLE II: Final model results on imdb sentiment analysis dataset.

| | |
|---|---|
| Test Loss | 0.2521449327468872 |
| Test Accuracy | 0.9054897427558899 |

The results presented were obtained by replacing the embedding layer with a pre-trained model from [59].

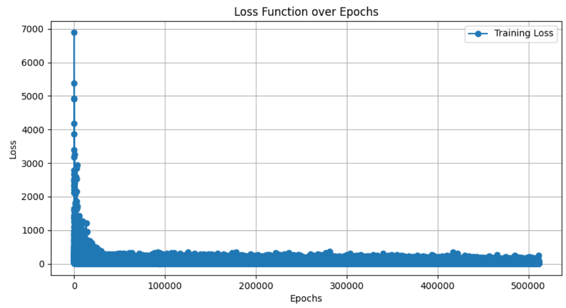
TABLE III: IMDb Sentiment Analysis Test Set Accuracy for Different Models in the Literature

| Model | Accuracy (%) | Reference |
|---|---|---|
| Naive Bayes (Baseline) | 83.5 | [60] |
| LSTM (Long Short-Term Memory) | 89.0 | [61] |
| BiLSTM with Attention | 91.2 | [62] |
| FastText | 88.5 | [63] |
| **RoBERTa+NAS Tree-based Classifier** | 90.5 | - |
| CNN (Convolutional Networks) | 90.6 | [64] |
| ULMFiT (Pretrained LSTM) | 94.0 | [65] |
| BERT-base (Fine-tuned) | 95.2 | [66] |
| RoBERTa (Fine-tuned) | 96.3 | [67] |
| DistilBERT | 95.1 | [68] |
| GPT-2 (Fine-tuned) | 95.0 | [69] |
| XLNet | 96.4 | [70] |
| ALBERT | 95.8 | [71] |
| ELECTRA | 96.6 | [72] |
| T5 (Text-to-Text Transfer) | 96.1 | [73] |
| GPT-3 (Few-shot) | 94.7 | [74] |
| DeBERTa (Fine-tuned) | 96.7 | [75] |
| ChatGPT (Prompting) | 96.0* | [76] |

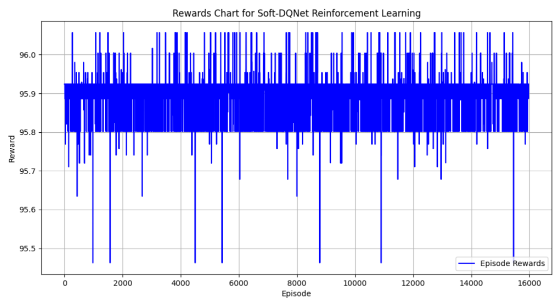
The neural architecture search task was performed using both language models: character-level using SoftQLearning and word-level using asynchronous advantage actor-critic training. In both cases, the problem's probability density function for each output was predicted by the models, and the final output is a result of a random experience with the model-predicted odds. This feature allows the models to represent more complex problems, such as NAS. This behavior also enables the model to learn the probabilistic aspects of a dataset; by fixing a serializable data ontology, it can generate datasets. Returning to the scope of this article, more specifically regarding these models' optimization capability in the NAS task, the learning and performance curves are presented below.

In the above experiment the model presented in [51], has two decoupled outputs which are used to compose the sequence - Network Structure Code. The Soft Q-Learning training method was adopted instead of DQNet-PL because the latest presents a very high training variance, making the training to not converge. Additionally the entropy regularization also helped to attain training convergence.

The transformer-based Vector Quantized Variational Auto Encoder (VQ-VAE) follows the same decoupling logic to compose the sequence, as described previously. In this case, the model has two outputs: the actor and the critic. The actor predicts odds for each possible model action, and the second output, the critic rates the overall model performance. In terms

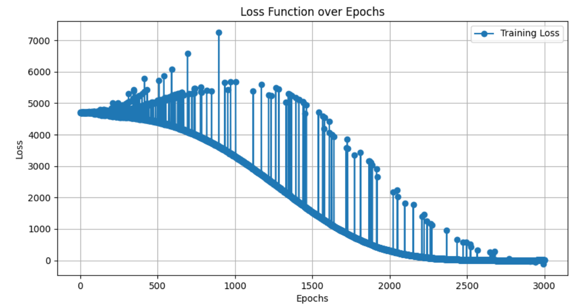



(a) Loss function of SoftQ-Learning using Char-Conv inspired architecture.

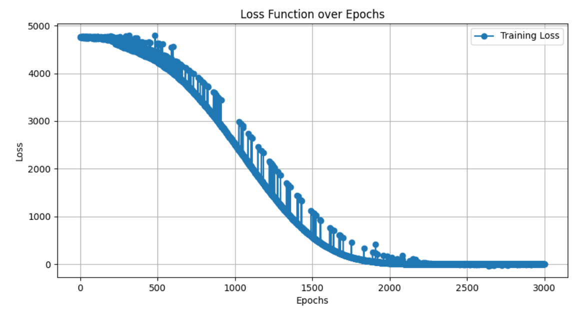



(b) RL reward function, child network's training accuracy, using SoftQ-Learning with Char-Conv inspired architecture.

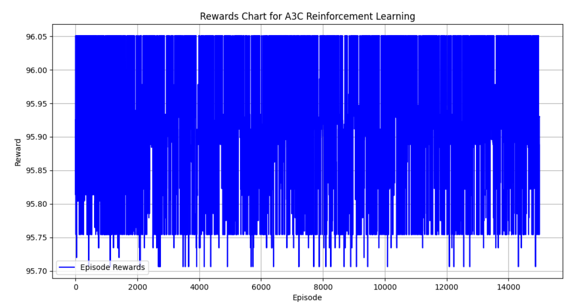
Figure 5: Comparison of loss and reward during training.




(a) Loss function of action sequence composing parameter during A3C training using Transformer-inspired architecture.




(b) Position loss function of A3C using Transformer-inspired architecture.




(c) RL reward and child network accuracy as functions of A3C using a Transformer-inspired architecture.

Figure 6: Training metrics during A3C using Transformer-inspired architecture: (a) Action sequence loss, (b) Position loss, and (c) Reward and accuracy.

of architecture, the actor-critic decoupling is made only in the model's last layer to shape the output according to the needs to generate the critic score and actor's odds.

Two Transformer-based VQ-VAE models were used to compose the sequence, one to generate the action and another to generate the position in the candidate sequence where the action value will be assigned. Bellow, the obtained training curves are presented:

The observable outliers are due to the epsilon-greedy technique used to introduce exploration in the algorithm's behavior.

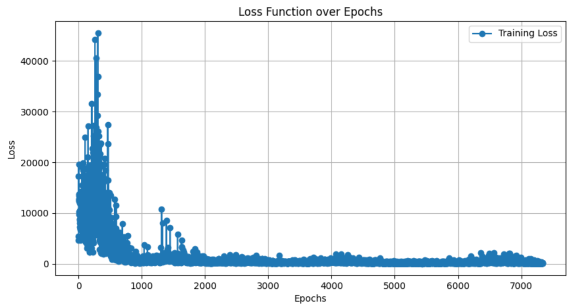
All the loss function plots in the presented results converge to zero, and the reward signals reflect the overfitting tendency of the proposed NAS methodology. The decoupling strategies are effective in stabilizing the training methodologies in both character and word-level approaches. Additionally, the sequence generalization and problem modeling capabilities are verified when observing the obtained training curves; both approaches exhibit stable behavior.

Next, the Traveling Salesman Problem results are presented. Experiments with both the architectures are presented bellow.

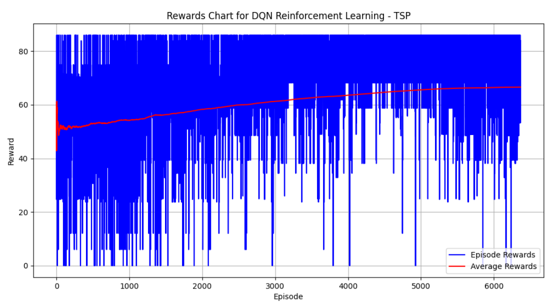
Starting from the Char-Conv as DQNet and as well as Policy network, the results were the following:

Both curves indicate that the RL agent is learning, as evidenced by the loss function's convergence to zero and the reward function's increasing behavior during training. Next, the VQ-VAE model is used in conjunction with A3C training to generate the salesman route:

The loss function chart exhibits zero convergence; therefore, training stability is concluded, and the generally increasing reward function reflects the VQ-VAE agent's learning. Depending on the problem complexity, generating action odds might be preferred rather than generating the agent's actions directly, as occurred with the Char-Conv architecture in NAS, where Soft-Q Learning was used, and in the TSP where DQNet-PL was utilized. The Transformer-inspired VQ-VAE demonstrates overall better training behavior compared to the Char-Conv architecture, as this model can map the search space into

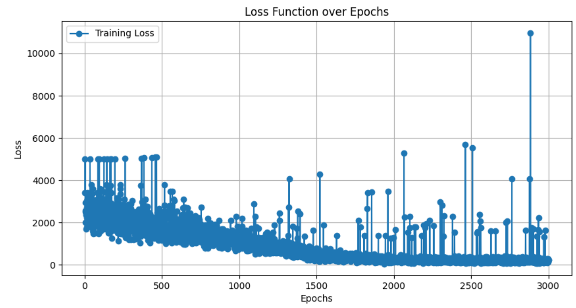



(a) Char-Conv architecture's loss function during DQNet-PL training, while solving the Traveling Salesman problem.

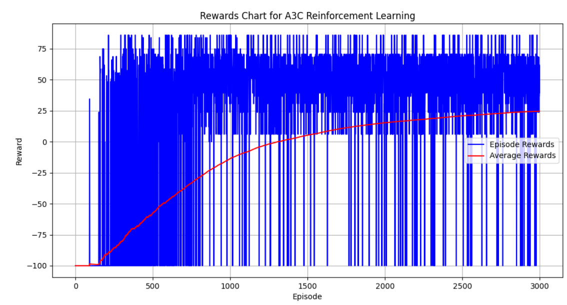



(b) Reward function of Char-Conv architecture during DQNet-PL training, while solving the TSP problem.

Figure 7: Training metrics of the Char-Conv architecture in DQNet-PL while solving the TSP: (a) Loss function and (b) Reward function.




(a) Loss function during A3C training using Transformer-inspired architecture in the TSP problem resolution.




(b) Reward function obtained by the VQ-VAE based agent in A3C.

Figure 8: Training metrics for TSP problem using A3C with Transformer-inspired architecture: (a) Loss function and (b) Reward function.

several sub-regions by utilizing the Vector-Quantized layer, thereby parallelizing the search.

## XIV. Error Analysis

During the experimentation phase of this work, the Traveling Salesman benchmark problem was addressed using A3C training together with the Transformer-based VQ-VAE model. Additionally, the Char-Conv model was tested alongside DQNet-PL training on the same problem. After several unsuccessful experiments resulted from the usual issues of high variance in the reward signal and a non-converging loss function, a functional decoupling methodology was developed and successfully applied to the TSP problem. The training results are presented in Figures 9 and 11 for the Char-Conv and VQ-VAE models, respectively.

In considering the NAS problem, the combination of Char-Conv with DQN-PL training did not succeed in solving this issue, as the loss function did not converge to zero. In contrast, the combination of VQ-VAE, A3C, and the respective decoupling effectively solved the problem (Figures 6 and 7). To address the limitations of solving the NAS problem using CharConv, SoftQ Learning with entropy regulation was employed, as it enables modeling the odds of each output and reduces the variance of the reward signal.

## XV. Conclusions and Future Work

Many problems are non-sequential and do not require strict left-to-right order dependency. To handle such cases, a value-position decoupling strategy is proposed. Considering the Transformer-based VQ-VAE trained with A3C, the model has two outputs: an actor output and a critic output. Instead of using two models, a single model is employed. In this way, the network weights are updated on both occasions: when the actor learns and when the critic learns. Two A3C models with a shared state and reward are used; one generates the new element's position, and the other generates the new element's values. The VQ-VAE architecture has the capability to divide the latent space into quantized subspaces and perform a parallelized search in each subspace. The deep convolutional network, trained using Deep Q-Learning for value generation and Soft Q-Learning for sequence generation, applies similar reasoning to design the training. One model with two outputs is responsible for generating the new element's position, while another model generates the new element's value. To make this training generative, the output odds are modeled, and the outputs are generated using a random experiment in which each output odd is defined by the Deep Q-models. Additionally, to reduce training variance, an entropy term is added to the loss function. This process is called entropy regularization and promotes training convergence toward zero. This study demonstrates that it is possible to generate sequences without causality constraints while still employing slightly adapted Reinforcement Learning techniques. Training convergence improves if the same model with two outputs is used to perform actions and critique its performance, regardless of its architecture.

Complex ontologies describing the candidate solutions can be encoded and serialized into integer sequences. The encoded sequences can then be optimized by this type of solver when used with a performance metric designed as the Reinforcement Learning reward. Since sufficient decoupling is achieved, the language models can absorb the problem's semantics and generate admissible candidate solutions of increasing quality. The position-value decoupling must be employed in the NAS scenario to avoid imposing causality in the sequence generation during the RL training. Additionally, using variational models in complex RL environments such as NAS is more efficient since they model the environment's unknown properties. The Transformer-based VQ-VAE is also capable of parallelizing the search due to the vector quantization layer. Looking toward the future, the models presented, along with the proposed training techniques, can be used to generate more than encoded solutions for a given problem. By selecting an appropriate reward function, the generated sequence can be utilized in the standard format to produce content similar to Generative Adversarial Networks. A comparison of the proposed solvers with other state-of-the-art heuristic search algorithms can be made to systematically explore the limitations of this proposal and extend its applicability domain. An analysis of the problem's degrees of freedom versus processing time will be conducted, focusing on solver quality analysis based on degrees of freedom, the solver's scaling with DoF, and the algorithm's parallelization. Going further in problem factorization into smaller problems can be automated, and the proposed learning techniques can be applied together with the parallelization capabilities of Vector Quantized layers using problem dissociation transforms such as the Laplace Transform. If a problem resolution process can be transposed into a computing tree, the currently generated n-ary tree for classifier construction can be reapplied into a problem resolution framework.

Regarding the algebraic properties of the Bellman operator, it is also possible to approach problem decomposition and simplification by decomposing the problem into Bellman blocks. Once these learnable blocks are identified, the proposed decoupling strategies can be applied to each block. This approach can be generalized, and the goal of searching for these blocks from given data can be replaced by the proposed NAS-based perspective.

## XVI. Acknowledgments

The English writing in this paper was reviewed by GPT-0 [77] and the mathematical formulations were Chat-GPT aided. We also thank Évora University and IARIA's reviewers for all the meaningful critiques.

## References


[1] C. Carvalho and P. Quaresma, "Heuristic search using language models and reinforcement learning," in *Proceedings of the Seventeenth International Conference on Information, Process, and Knowledge Management (eKNOW 2025)*. IARIA, 2025, pp. 1–12, © IARIA, 2025.

[2] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[3] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," *arXiv preprint arXiv:1401.4082*, 2014.

[4] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations*, 2017.

[5] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in Neural Information Processing Systems*, 2014, pp. 3581–3589.

[6] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, "Understanding disentangling in beta-vae," *arXiv preprint arXiv:1804.03599*, 2018.

[7] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in Neural Information Processing Systems*, 2015, pp. 3483–3491.

[8] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," *arXiv preprint arXiv:1606.03657*, 2016.

[9] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[10] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," *arXiv preprint arXiv:1505.05770*, 2015.

[11] N. Dilokthanakul, P. A. Mediano, M. Garnelo, M. C. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan, "Deep unsupervised clustering with gaussian mixture variational autoencoders," *arXiv preprint arXiv:1611.02648*, 2016.

[12] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schölkopf, "Wasserstein auto-encoders," *arXiv preprint arXiv:1711.01558*, 2017.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[14] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *International Conference on Learning Representations*, 2016.

[15] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[16] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, 2017, pp. 5767–5777.

[17] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4401–4410.

[18] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[19] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[20] H. Ishfaq, A. Hoogi, and D. Rubin, "TVAE: Triplet-Based Variational Autoencoder using Metric Learning," no. 2015, pp. 1–4, 2018. [Online]. Available: http://arxiv.org/abs/1802.04403

[21] S. Paul, "Vector-Quantized Variational Autoencoders," 2021. [Online]. Available: https://keras.io/examples/generative/vq_vae/

[22] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975.

[23] D. E. Goldberg, "Genetic algorithms in search, optimization, and machine learning," in *Addison-Wesley*, 1989.

[24] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," *Doctoral dissertation, University of Michigan*, 1975.

[25] J. H. Holland, "Building blocks, fringe search, and genetic algorithms," in *Foundations of Genetic Algorithms*, vol. 1, 1992, pp. 1–8.

[26] M. Mitchell, "An introduction to genetic algorithms," *MIT Press*, 1996.

[27] J. R. Koza, "Genetic programming: On the programming of computers by means of natural selection," in *MIT Press*, 1992.

[28] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[29] D. E. Goldberg and K. Deb, "Comparative analysis of selection schemes used in genetic algorithms," *Foundations of Genetic Algorithms*, vol. 1, pp. 69–93, 1991.

[30] K. Deb, "Multi-objective optimization using evolutionary algorithms," *Wiley*, 2001.

[31] T. Bäck, "Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms," *Oxford University Press*, 1996.

[32] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[33] J. L. Elman, "Finding structure in time," in *Cognitive Science*, vol. 14, 1990, pp. 179–211.

[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[35] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI preprint*, 2018.

[36] 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.

[37] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[38] Y. Goldberg, "A primer on neural network models for natural language processing," *Journal of Artificial Intelligence Research*, vol. 57, pp. 345–420, 2016.

[39] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[40] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

[41] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[42] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[43] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[44] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado *et al.*, "Google's multilingual neural machine translation system: Enabling zero-shot translation," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 339–351, 2017.

[45] C. Chu, R. Wang, and R. Dabre, "A survey of domain adaptation for neural machine translation," *arXiv preprint arXiv:1806.00258*, 2018.

[46] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pp. 1–16, 2017.

[47] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive Neural Architecture Search," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11205 LNCS, 2017, pp. 19–35.

[48] T. Shen, A. Kuncoro, X. Liu, O. Firat, P. Barham, and Q. V. Le, "Minimum risk training for neural machine translation," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016, pp. 1689–1699.

[49] 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. 2017-December, no. Nips, pp. 5999–6009, 2017.

[50] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural discrete representation learning," *arXiv preprint arXiv:1711.00937*, 2017.

[51] X. Zhang and Y. LeCun, "Text Understanding from Scratch," 2015. [Online]. Available: http://arxiv.org/abs/1502.01710

[52] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[53] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations (ICLR)*, 2016.

[54] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*. PMLR, 2016, pp. 1928–1937.

[55] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *International Conference on Machine Learning (ICML)*. PMLR, 2017, pp. 1352–1361.

[56] C. Szegedy, S. Reed, P. Sermanet, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," pp. 1–12.

[57] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019. [Online]. Available: http://arxiv.org/abs/1907.11692

[58] S. Tripathi, R. Mehrotra, V. Bansal, and S. Upadhyay, "Analyzing sentiment using imdb dataset," in *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*, 2020, pp. 30–33.

[59] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. [Online]. Available: http://www.aclweb.org/anthology/P11-1015

[60] B. Pang and L. Lee, "Thumbs up? sentiment classification using machine learning techniques," in *Proceedings of the ACL*. Association for Computational Linguistics, 2002.

[61] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification," in *Proceedings of the 2015 conference on empirical methods in natural language processing (EMNLP)*, 2015, pp. 1422–1432.

[62] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," *Proceedings of ACL*, 2016.

[63] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, vol. 2, 2017, pp. 427–431.

[64] R. Johnson and T. Zhang, "Deep pyramid convolutional neural networks for text categorization," *ACL*, 2017.

[65] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 328–339.

[66] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[67] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[68] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2020.

[69] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, 2019.

[70] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

[71] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2020.

[72] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, “Electra: Pre-training text encoders as discriminators rather than generators,” in *ICLR*, 2020.
[73] 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,” *JMLR*, 2020.
[74] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–1901.
[75] P. He, X. Liu, J. Gao, and W. Chen, “Deberta: Decoding-enhanced bert with disentangled attention,” *ICLR*, 2021.
[76] OpenAI, “Gpt-4 technical report,” *OpenAI Technical Reports*, 2023.
[77] (2025) Zerogpt – freegrammar checker & spell checker ai-powered. Accessed: 2025-07-13. [Online]. Available: https://www.zerogpt.com/grammar-checker?r=gwords&gad_source=1&gad_campaignid=21008842773&gclid=Cj0KCQjwxJvBBhDuARIsAGUgNfiL-gX33yP9aHgXBh39K6i8q45ykTzyznD-uFuDr_pZs-5L7rivI9MaAhjZEALw_wcB