NEURAL-GUIDED LEARNING OF INTEGER SEQUENCES

Jaden Long, Tony Wu, Dennis Xu
Duke University (Names in alphabetical order)
{yl708, sw478, yx188}@duke.edu

ABSTRACT

In this project, we adapt neural-guided search methods to tackle the problem of inferring the latent generating functions underlying integer sequences. We implement brute force search and simple seq2seq and fine-tuned GPT-3 as our baselines, and propose a neuro-symbolic approach based on Monte Carlo Tree Search (MCTS) that incorporates symbolic function rules to iteratively train a neural policy network as our heuristic. We observed that our MCTS-based neural symbolic method outperformed all other methods across all experiments we conducted. To the best of our knowledge, our project is the first to date that surveys neural symbolic methods for solving integer sequences. All of the code used in this project can be found at https://github.com/denx20/CS-590-Project.

1 INTRODUCTION

Integer sequences are ubiquitous and often beautiful. The most famous integer sequence is probably the Fibonacci sequence, where each term is given by the sum of the previous two terms, with the starting terms 0 and 1. The Fibonacci sequence is often adjoined with epithets such as "fabulous" (Posamentier, 2007) and "beautiful" (Campbell, 2010).

In this paper, we explore computational approaches to identify the underlying generating function f of an integer sequence s given the first few terms of the sequence. As an example, given an observation of the Fibonacci sequence

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...$$

the goal of our algorithm would be to identify the underlying generating function, $f(n) = f(n-1) + f(n-2)$, or any other equivalent function that matches this function at every n. In this project, we restrict f to be a polynomial with integer coefficients. We also add restrictions on s and the complexity of f to prevent overfitting, which is possible with highly nonlinear interpolation methods such as polynomial regression and smoothing splines.

While there has been existing work on inferring underlying functions from real-valued inputs and outputs (described in more details in section 2), our work is the first work known to date that surveys methods for integer sequences. The main difficulty of directly applying machine learning approaches to this problem arises from the integer constraint. On one hand, most quantitative learning approaches are only capable of modeling response variables that are continuous. Although methods like Poisson regression do exist for discrete predictions, their assumptions are often too constrained to be useful for many real-world scenarios. On the other hand, given the limited size of our input, it is also not clear how we could best encode this to provide the most relevant information for standard deep learning based techniques. As a result, the reasoning underpinning all of our approaches stems from this discreteness of the integers.

1.1 BACKGROUND AND TERMINOLOGY

Problem definition. For some underlying function $f : n, a_{n-s}, a_{n-s+1}, ..., a_{n-1} \rightarrow a_n,\ s, n \in \mathbb{Z}^+,\ n > s$ with integer coefficients and an initial set of numbers $b_1, ..., b_s \in \mathbb{Z}$, we define the

infinite sequence $\{a_n\}_{n=1}^{\infty}$ such that

$$a_n = \begin{cases} b_n, & \text{if } n \leq s \\ f(n), & \text{otherwise.} \end{cases}$$

Our problem is the reverse: given the first few terms $\{a_n\}_{n=1}^{k}$, we wish to find f^* such that $f^*(n) = f(n)$ for all positive integers $n > s$.

Terminology. We define several relevant terms used throughout the paper:

- "Function", or "formula", refers to the generating function of the sequence which takes in an index n and outputs the element at that index (a_n), e.g. $f(n) = 2f(n-1) + 3n$.
- We will refer to each term in the function, without specification for its coefficients, as "term", e.g. $f(n-1)$ and n are the two terms in the previous function.
- We will refer to the coefficients of each term in the function as "coefficients", e.g. 2 and 3 are the coefficients for the two aforementioned terms.
- Given a list of all possible terms to include in any function, we indicate which terms are used in a specific function via a "binary mask". For example, if the list of all possible terms is $[1, n, n^2, f(n-1)]$, then the function $f(n) = 2f(n-1) + 3n$ will have the binary mask [0, 1, 0, 1] and the function $f(n) = n + 5$ will have the binary mask [1, 1, 0, 0].
- The root mean square error (RMSE) quantifies the difference between the ground truth sequence and some predicted sequence. Given the ground truth function $f(n)$ and a predicted function $f^*(n)$, the RMSE of the first k terms is defined as $\sqrt{\sum_{n=1}^{k} \left(f(n) - f^*(n)\right)^2}$.

1.2 Domain-Specific Language

Since the set of all possible underlying functions that generate integer sequences is infinite, we restrict our task to a clear and tractable subset of the functions by defining a Domain-Specific Language (DSL) that specifies which types of function terms are allowed. More specifically, we will focus on functions that satisfy all of the following assumptions:

> *Assumption 1.* $f(n)$ can be expressed as a linear combination of the monomials $n, f(n-1), f(n-2), f(n-3)$ of degree at most 4, and any expression involving earlier terms (i.e. $f(n-4)$) must have an equivalent expression consisting only of combinations of these terms. Interaction terms are allowed as long as they can be written as products of two of these terms.
>
> *Assumption 2.* In the expression of $f(n)$, coefficient of each term must be an integer in the range $[-5, 5]$.
>
> *Assumption 3.* In the expression of $f(n)$, the number of terms with nonzero coefficients is either 2 or 3.

While these assumptions may appear restrictive, we can avoid overfitting input sequences of length k with degree-$(k-1)$ polynomials (through Assumption 1) and ensure that the sequences generated satisfy the integer constraints whenever the initial terms happen to be integers (through Assumption 2). Moreover, our DSL still allows the existence of equivalent expressions. For instance, the function $f(n) = \frac{1}{2}n(n+1)$ has two equivalent forms in our DSL, namely $f(n) = f(n-1) + n$ and $f(n) = n^2 - f(n-1)$.

2 Related Work

Integer Sequences. One popular database for integer sequences is the On-Line Encyclopedia of Integer Sequences (OEIS) created by Neil J. A. Sloane. Sloane first published *A Handbook of Integer Sequences* through Academic Press in 1973 during his time at AT&T Bell Labs, which then contained 2372 sequences in lexicographic order. OEIS now contains over 358,000 integer sequences. (Sloane & Inc., 2022)

Symbolic Regression. There has been a few works on symbolic regression, which solves the task of finding symbolic functions that underpin empirical observations. In a seminal paper published on Science, Schmidt & Lipson (2009) described a way to use genetic algorithms to infer natural laws from experimental data. Since then, there has been a large number of papers on uncovering natural laws based on physical observations (Rudy et al., 2017) (Cornelio et al., 2021) (Xie et al., 2021). In particular, there exists an open-source implementation of the genetic-algorithm based approach from the 2009 paper in Python and Julia via the package PySR (Cranmer, 2020). However, because PySR is designed for regression, it does not support recurrence relations, which are essential to the types of integer sequences we are interested in.

Neuro-Symbolic Machine Learning. Many recent works on neuro-symbolic machine learning focus on program synthesis given a specific DSL. For instance, CrossBeam is a bottom-up search method proposed by (Shi et al., 2022) that synthesizes new programs by training a neural network to combine explored programs into new programs, given the search history and outcomes of partial program executions. More similar to ours is the work done by (Biggio et al., 2021), which applies neural network based symbolic regression to find the underlying equation from a set of input-output pairs.

In our work, we apply neuro-symbolic techniques to find the underlying generating formula for integer sequences. The task itself is a natural extension of those tackled by the previously-studied symbolic regressions, which usually do not include recurrence relations. Our task resembles program synthesis in the sense that we seek to find an expression $f^*(n)$ for which $f^*(n) = f(n) = a_n$ for $n = 1, \ldots, k$ given the input sequence $\{a_n\}_{n=1}^{k}$. Compared with other methods that only provide numerical predictions for the next numbers in the sequence, our objective of finding the actual formula lends itself to more robust and verifiable solutions.

3 Approach

3.1 Brute Force Search

A first natural approach is to brute force search the space of all possible functions as defined by our DSL. Given the lists of allowed terms $\boldsymbol{t}$ ($|\boldsymbol{t}| = n$) and coefficients $\boldsymbol{c}$ ($|\boldsymbol{c}| = m$), some $1 < k \leq n$, and a target sequence $\boldsymbol{s}$ ($|\boldsymbol{s}| = h$), here is our implementation:

Algorithm Brute Force Search

Require: $\boldsymbol{t}, \boldsymbol{c}, \boldsymbol{s}, k$
- $combs \leftarrow$ all k-combinations of $\boldsymbol{t}$ ▷ $O({}_nC_k)$
- $perms \leftarrow$ all possible coefficient arrangements ▷ $O(m^k)$
- **for** $comb$ in $combs$ **do**
  - **for** $perm$ in $perms$ **do**
    - $f \leftarrow$ apply $perm$ to each term in $comb$ ▷ $O(k)$
    - $curr \leftarrow$ compute sequence based on f ▷ $O(h)$
    - $rmse \leftarrow$ compute RMSE between $curr$ and $\boldsymbol{s}$ ▷ $O(h)$
    - **if** $rmse = 0$ **then**
      - **return** f
    - **else**
      - **continue**
    - **end if**
  - **end for**
- **end for**

Note that according to our constraints, $k < h$, so in the worst case, the runtime of this approach is $O\left({}_nC_k \cdot m^k \cdot h\right) \approx O\left(\left(\frac{m \cdot n}{k}\right)^k \cdot h\right)$, which grows exponentially fast as we increase n and quickly becomes infeasible. The infeasibility of brute force search motivates better heuristics for solving this problem.

3.2 SIMPLE SEQ2SEQ

Our seq2seq model takes the sequence of integers as input and outputs the binary mask of the function generating that sequence. Code for our implementation is adapted from https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html, with an RNN encoder and attention RNN decoder. The exact architecture of each of these components are shown in A.2. See `seq2seq.ipynb` for details.

3.3 FINE-TUNED GPT-3

GPT-3 is a large language model developed by OpenAI that is based on the stacking of transformers (Vaswani et al., 2017) (Brown et al., 2020). Before fine-tuning, we first formatted our prompts according to the guidelines provided by OpenAI (more on this in section 4.2). Then, we performed fine-tuning using different pretrained variants of the model (code-named ada, babbage, curie, davinci, in order of increasing number of parameters) as provided by the OpenAI API (https://openai.com/api/). See `GPT3 Finetune.ipynb` for the exact implementation details.

3.4 NEURAL POLICY GUIDED MCTS

In this subsection we introduce our neuro-symbolic approach to solving the integer sequence task, incorporating symbolic rules about polynomials and recurrence relations while training our neural network model.

First, we define our search space as follows: the search space is a search tree, rooted at the trivial function expression $f(n) = 0$. Each node in the search tree contains a set of allowed terms S in our DSL, representing the class of function expressions consisting entirely of the terms in S with nonzero coefficients in $[-5, 5]$. A child node S' of S can be obtained by adding a new term to S. A terminal node is arrived when a $\langle EOS \rangle$ token is appended or when the number of terms exceeds a pre-specified limit, which we set to 4 since in our DSL the number of terms in a generating function is 2 or 3. The symbolic rules we impose on the search tree are

Inspired by AlphaGo (Silver et al., 2017), we develop a Monte Carlo Tree Search-based search algorithm, where we use a neural network model to predict (1) the policy and (2) reward value of the nodes based on (1) the integer sequence and (2) current node's terms, and the neural network is iteratively trained with training examples gathered from the MCTS search process. More specifically, our algorithm works as follows:

1. Search Step:
   - For each training sequence, initialize an empty search tree and perform MCTS, where the initial guess of the policy and reward of newly-explored node is determined by the neural network. The remaining part of the tree exploration is the same as the standard MCTS algorithm, with reward at terminal nodes assigned as $10 - depth$ if the terminal node is a correct solution, else $-RMSE$.
   - When a terminal node is reached, collect examples from randomly-sampled search tree nodes, in the format of (sequence, node terms, policy after MCTS update, reward after MCTS update).
2. Train Step:
   - Using the collected training examples, train the neural network's predicted node policy and reward to be close to the policy and reward obtained from MCTS. The loss function at a node u is
     $$\mathcal{L}(u) = \left[\text{reward}_{\text{NN}}(u) - \text{reward}_{\text{MCTS}}(u)\right]^2 + BCE(\text{policy}_{\text{NN}}(u), \text{policy}_{\text{MCTS}}(u))$$
     where $BCE()$ refers to the Binary Cross Entropy loss.
3. Repeat.

We intentionally design the reward at terminal node to include a $-depth$ term to discourage overfitting expressions, which tend to consist of more function terms from the DSL than necessary. We also use $RMSE$ as a surrogate to estimate how far the class of function represented by the terminal node deviates from the given integer sequence.

4 EXPERIMENTS

4.1 DATASETS AND EVALUATION

4.1.1 DATASET GENERATION

Our synthetic dataset is generated using functions that conform to our assumptions listed in section 1.2. Based on our list of allowed terms (refer to A.1 for the full lists with and without interaction terms), we first sample a random k-combination of the terms, denoted as $\{\text{term}_i\}_{i=1}^{k}$ and represented as a boolean mask over the possible terms. Then, for each term_i, we randomly choose an integer coefficient in the range $[-5, 5]$, resulting in the function defined by $f = \sum_{i=1}^{k} c_i \cdot \text{term}_i$. Using this function, we generate its corresponding sequence $\{a_n\}_{n=1}^{7}$, with initial terms chosen uniformly in the set $\{1, 2, 3\}$ as needed. Hence, given a sequence bound b and k, our dataset consists of tuples of the form $(\mathbf{X}, y) = (\{a_n\}_{n=1}^{7}, boolmask)$ where $\max\left(|\{a_n\}_{n=1}^{7}|\right) < b$ and exactly k values in the boolean mask $boolmask$ are true. Using $b = 2000$ for the dataset with interaction terms and $b = 1000$ for the dataset without, we generate 1000 such tuples for each of $k = 2, 3$ and randomly split them into sets of sizes 800 and 200 for training and test respectively.

4.1.2 EVALUATION

We first note that we cannot simply check for equality of the boolean masks since there may exist multiple equivalent expressions in our DSL that fit the given sequence perfectly, and we would consider all of them as valid solutions. For instance, for the input sequence given by $1, 3, 6, 10, 15, 21, 28$, both $f^*(n) = n + f^*(n-1)$ and $f^*(n) = n^2 - f^*(n-1)$ are correct. Therefore, we primarily evaluate the models based on the following two metrics:

Mean RMSE. Given the predicted boolean mask for a sequence v in our test set V, we first grid search over all valid coefficients to form all possible functions based on the chosen terms. Then, for each function, we compute the RMSE between its corresponding sequence and the target sequence. We define RMSE_v as the minimum RMSE achieved over all such sequences generated by the possible functions. Finally, we take the average of this over the entire test set V to get the mean RMSE:

$$\text{Mean RMSE} := \frac{1}{|V|} \sum_{v \in V} \text{RMSE}_v.$$

Percentage Correct. For this metric, we simply compute the percentage based on the number of test sequences $v \in V$ for which $\text{RMSE}_v = 0$.

4.2 GPT-3 BASELINE

Following the guidelines provided by OpenAI, we first appended "->" to the end of all the input sequences for better indication of prompting. Then, we also appended an $\langle EOS \rangle$ token to the end of the boolean masks to signal for the completion of the answer to the prompt. Before training, we further split the prepared training set of size 800 into 640 for training and 160 for validation. Based on cost considerations, we used all the default hyperparameters (listed in A.4) without further hyperparameter tuning.

4.3 HUMAN BASELINE

The three authors and another friend were given five random problems from the $nterms = 2$ with no interaction term dataset for 10 minutes.

After ten minutes have elapsed, the participants have only figured out one out of the five problems (the underlying function for which happens to only have one term) and had no lead on any of the other sequences. Hence we decided that $nterms = 3$ would not even be worth trying.

4.4 MCTS MODEL AND TRAINING DETAILS

Neural network architecture. The neural network model used consists of an encoder and a decoder. The encoder is a standard transformer encoder, with hidden dimension $d \in \{64, 256, 512, 1024\}$,

and the decoder is implemented as one of MLP, GRU, or transformer decoder, with the same hidden dimension as the encoder and number of layers between 4 and 6, inclusive. Two projection heads, one for reward prediction and one for policy prediction, are added on top of the last layer of the decoder.

Training details. During training, we repeat the search-train process for 5 times, and in the search step, for each training sequence we reinitialize the search tree and perform MCTS for 3 times before collecting the training examples in order to reduce the likelihood of missing a correct terminal node, since correct solutions are sparse in our search space and hard to find. In the train step, we split the examples collected from MCTS into a training set and a validation set 9-to-1. We use the Adam optimizer with initial learning rate 10^{-4} or 5×10^{-5}, and reduce the learning rate by half when the validation loss gets stuck on a plateau.

4.5 RESULTS

We compare our neuro-symbolic approach against seq2seq and GPT-3 baselines in all the experiment settings. From Table 1 to 4, it can be observed that MCTS-guided neuro-symbolic learning can outperform all baselines, even though it does not require a larger neural network size (i.e. the size of GPT-3). Moreover, despite the increase in difficulty when the number of allowed terms doubles, our approach does not show a drastic drop in performance, which demonstrates robustness to scaling of DSL size.

nterms=2, # DSL terms=12	Correctness	RMSE
Human Baseline	20.0%	N/A
seq2seq	12.0%	126.1
GPT3-ada	18.0%	40.1
GPT3-babbage	22.5%	25.0
GPT3-curie	25.5%	28.1
GPT3-davinci	30.5%	23.5
MCTS-MLP (Ours)	26.0%	21.8
MCTS-GRU (Ours)	**33.0%**	**15.7**
MCTS-Transformer (Ours)	32.8%	16.1

Table 1: Results on $nterms = 2$ integer sequences, with 12 allowed terms in DSL (i.e. no interaction terms allowed). RMSE of seq2seq was not calculated because the seq2seq model can output a large number of terms even in the case where the number of terms in the underlying function is small. The humans gave up and did not attempt any further experiments because we failed miserably at even $nterms = 2$.

nterms=3, # DSL terms=12	Correctness	RMSE
seq2seq	12.5%	714.1
GPT3-ada	9.0%	334.7
GPT3-babbage	11.5%	215.0
GPT3-curie	11.0%	119.6
GPT3-davinci	9.0%	284.0
MCTS-MLP (Ours)	19.8%	9.9
MCTS-GRU (Ours)	16.5%	11.5
MCTS-Transformer (Ours)	**21.5%**	**7.9**

Table 2: Results on $nterms = 3$ integer sequences, with 12 allowed terms in DSL (i.e. no interaction terms allowed).

5 ANALYSIS

It came as no surprise that Monte Carlo Tree Search-based neuro-symbolic learning performed the best across the board in both RMSE and correctness, under all three architectures, and outperformed

nterms=2, # DSL terms=24	Correctness	RMSE
seq2seq	1.0%	79.6
GPT3-ada	7.5%	6018.7
GPT3-babbage	8.5%	3443.9
GPT3-curie	10.5%	1265.7
GPT3-davinci	9.5%	864.0
MCTS-MLP (Ours)	**37.5%**	13.8
MCTS-GRU (Ours)	23.0%	14.5
MCTS-Transformer (Ours)	29.5%	**12.5**

Table 3: Results on $nterms = 2$ integer sequences, with 24 allowed terms in DSL (i.e. interaction terms allowed)

nterms=3, # DSL terms=24	Correctness	RMSE
seq2seq	0.0%	1234.4
GPT3-ada	2.5%	12997.1
GPT3-babbage	3.5%	5464.3
GPT3-curie	4.0%	5656.3
GPT3-davinci	2.5%	782.5
MCTS-MLP (Ours)	13.5%	25.7
MCTS-GRU (Ours)	11.0%	**18.1**
MCTS-Transformer (Ours)	**16.0%**	21.9

Table 4: Results on $nterms = 3$ integer sequences, with 24 allowed terms in DSL (i.e. interaction terms allowed)

all other approaches by a significant margin. We believe one main reason is that the tree search process offers rich information (even more informative than explicit ground-truth labels of which terms are used) about the which function terms in the DSL "resembles" the input integer sequence the most, and which function still terms are missing conditioned on the previously-chosen terms and the original sequence. To this end, architectural choice of the neural network used in MCTS comes as a secondary determining factor for performance.

The significance of the success of our neuro-symbolic method is twofold. First, we show that incorporating symbolic rules about the underlying generating functions of integer sequences can boost neural network performance on a difficult task with multiple constraints like solving integer sequences. Second, we show that even in a search tree where positive rewards are very sparse, MCTS can provide abundant information about task that can be used for successful on-policy training of the neural network, which effectively reduces the reliance on explicit supervision.

6 Conclusion

In this project, we explored different models for symbolic regression on integer sequences. Overall, our neuro-symbolic approach based on Monte Carlo Tree Search is an apt model for the task of integer sequence symbolic regression. Through tackling this task, we shed light on the capability of neuro-symbolic learning of tasks with constraints and highlight the benefits of enforcing symbolic rules in training neural networks.

Further exploration of this subject could include adding recurrence terms to a conventional symbolic regression framework and testing the efficacy of conventional symbolic regression. In the future, one can also expand the expressiveness of our DSL by adding operations specific to integers, such as reading each number from right to left (e.g. entry A002942 in OEIS). Another immediate open problem for future work is to test the robustness of our approach on the collection of "unsolved" integer sequences on OEIS.

AUTHOR CONTRIBUTIONS

- Jaden: Sequence and function generation, symbolic regression, seq2seq
- Dennis: Sequenece and function generation, GPT-3
- Tony: Monte Carlo Tree Search-based neuro-symbolic learning

REFERENCES

Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pp. 936–945. PMLR, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Sarah C Campbell. *Growing patterns*. Boyds Mills Press, Honesdale, PA, March 2010.

Cristina Cornelio, Sanjeeb Dash, Vernon Austel, Tyler R. Josephson, Joao Goncalves, Kenneth L. Clarkson, Nimrod Megiddo, Bachir El Khadir, and L. Horesh. Ai descartes: Combining data and theory for derivable scientific discovery. 2021.

Miles Cranmer. Pysr: Fast & parallelized symbolic regression in python/julia, September 2020. URL http://doi.org/10.5281/zenodo.4041459.

Alfred S Posamentier. *The fabulous Fibonacci numbers*. Prometheus Books, Amherst, NY, May 2007.

Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science advances*, 3(4):e1602614, 2017.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

Kensen Shi, Hanjun Dai, Kevin Ellis, and Charles Sutton. Crossbeam: Learning to search in bottom-up program synthesis. *arXiv preprint arXiv:2203.10452*, 2022.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Neil J. A. Sloane and The OEIS Foundation Inc. The on-line encyclopedia of integer sequences, 2022. URL http://oeis.org/?language=english.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Xiaoyu Xie, Wing Kam Liu, and Zhengtao Gan. Data-driven discovery of dimensionless numbers and scaling laws from experimental measurements. 2021.

A APPENDIX

A.1 DOMAIN-SPECIFIC LANGUAGE FOR SEQUENCE-GENERATING FUNCTION

We list the allowed terms in our DSL, which are the building blocks for the generating function that underlies integer sequences in our paper.

- Regular terms: $1, n, n^2, n^3, f(n-1), f(n-1)^2, f(n-2), f(n-2)^2, f(n-3), f(n-3)^2$

- Interaction terms: $f(n-1)f(n-2), f(n-1)^2f(n-2), f(n-1)f(n-2)^2, f(n-1)^2f(n-2)^2, f(n-1)f(n-3), f(n-1)^2f(n-3), f(n-1)f(n-3)^2, f(n-1)^2f(n-3)^2, f(n-2)f(n-3), f(n-2)^2f(n-3), f(n-2)f(n-3)^2, f(n-2)^2f(n-3)^2$

With $\langle SOS \rangle$ and $\langle EOS \rangle$ tokens included, our DSL contains 12 possible function terms when interaction terms are forbidden and 24 possible function terms when interaction terms are allowed.

A.2 Architecture of the simple seq2seq model

See figure 1.

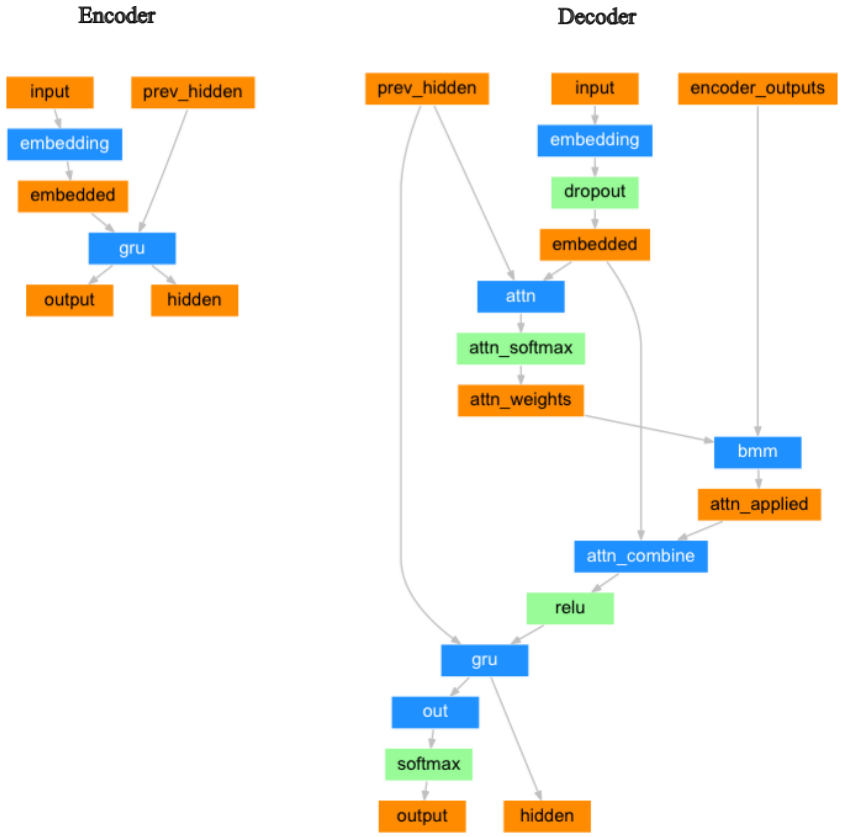

Figure 1: Architecture of the simple seq2seq model. Figure adapted from https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html.

A.3 Training Details of the Simple seq2seq Model

The start-of-string and end-of-string tokens are set to 10001 and 10002, respectively. Because of a bug in the program, training and testing dataset is directly generated using `sequence_generator.make_n_random_functions()`. The model takes one step after seeing one input-output pair, and for each trial, 20,000 training steps are made. Because seq2seq can output an unlimited number of tokens before the end-of-string token, they are cropped up to the nterms value for that trial. Details can be found in `seq2seq.ipynb`.

A.4 Hyperparameters in GPT-3 Fine-tune

Here are the default hyperparameters we used for fine-tuning all the GPT-3 baselines:

n_epochs = 4, batch_size=1, learning_rate_multiplier=0.05.

Terms in f	Average RMSE	Terms Correct	NN Parameters
2	53.75724089939156	31	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	16.255148488097277	47	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	82.93421091773433	14	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	13.47031803517485	49	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	66.8318371922878	10	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	38.3420775689672	28	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	11.3902964891051	52	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	17.525671998119076	53	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	23.88987559352432	38	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	14.3121288887179O2	48	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4"
2	39.32581252964236	13	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	81.36011801694045	19	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	66.28085991391036	21	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	17.68128930271921	49	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4"
2	9.73326849150304	49	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4"
2	66.59810345321638	11	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	17.931376296942577	47	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4"
2	79.95533065305042	10	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	70.17694243792009	11	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4"
2	71.50461603838241	14	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	60.371141739232925	11	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	64.5914029735688	27	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	53.2096317968788	11	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	39.6532507244042	45	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4"
3	36.79559971857854	14	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	46.12244473826806	11	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	84.92647286348236	14	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	85.20588020431303	10	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	41.779395598318686	8	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4"
2	36.09852224299059	16	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	34.1849921304523	13	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	30.875601391483272	13	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	16.6932734332517	25	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4"
3	76.33874834419507	11	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	20.67674307779223	18	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4"
3	106.65611438992356	8	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	59.23853254380655	22	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4"
2	61.60113453678044	11	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4"
3	74.83481269972803	4	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	32.73440308014921	31	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4"
3	66.97688267360088	5	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	68.58572153823945	16	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	40.017053827995085	11	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4"
2	100.41345961533744	16	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	94.60020880585338	11	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	56.78079351870877	29	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4"
3	41.43175822355042	11	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	79.96304343907545	9	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	71.81964730602076	9	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 4"
3	45.96905440173156	5	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4"
3	59.73580333746192	7	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4"
3	103.10217912652162	13	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	87.66410295938527	10	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	61.2368043538558	14	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	60.323353970065455	7	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	99.58724606549713	6	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4"
2	48.82717245023989	18	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4"
2	89.99650517418401	12	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	93.36677225528564	5	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	102.19687606688883	6	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	65.70900380065281	7	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 4"
3	114.91994827530776	8	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	74.79310081882876	13	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4"
3	68.23521183699789	10	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	93.08504249966798	6	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	86.53095092784581	9	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	113.473127098019	4	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4"
3	40.47167437291329	15	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4"
3	136.85866464926218	8	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"

Table 5: MCTS Hyperparameter Sweep Results with Interaction Terms

A.5 MCTS Hyperparameter Sweep

With interaction terms: See tables A.5 and 6.

Terms in f	Average RMSE	Terms Correct	NN Parameters
2	21.74138072116308	36	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	28.47997050255763	35	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	14.49960140910298	46	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	29.779693371406186	19	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	15.678338832617504	48	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	17.021167422578337	23	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	15.420896194185682	41	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	11.881430420326655	72	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	13.182996304637705	51	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5"
2	58.99038317040702	0	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	12.832912377515973	38	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	24.264040044547613	21	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	12.01694256258872	57	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5"
2	13.165416656576877	54	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6"
2	19.25790138555037	30	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	19.3060829263874	27	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	69.55917129248729	3	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	17.263062465302404	44	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5"
3	35.827444799111625	9	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	13.779905582541804	75	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	14.63671754017116	44	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	12.847930113561915	47	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	9.684159762337204	58	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6"
2	16.49903781108282	43	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	14.185213922089872	39	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	16.374676229981986	54	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	11.792406904786707	53	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6"
2	13.788924014359006	49	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6"
2	12.198337981308766	57	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5"
3	85.88179701287184	10	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	16.183315543981607	43	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	13.688267317117848	36	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5"
3	77.695608036925	11	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	11.895308083193369	55	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6"
2	12.50665880619866	59	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5"
2	16.684385295672364	40	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	13.397498365571174	45	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	26.47202666513626	7	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	33.06961559139317	16	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	78.0026567869823	5	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	21.668493438048163	17	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5"
3	82.44186751091894	8	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	22.343858047595603	14	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	18.084019147466623	22	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	25.68423869042213	27	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	29.48585557248497	19	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	18.818204496384453	22	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5"
3	72.03382352178339	12	"'embed˙size': 1024, 'hidden˙size': 1024, 'num˙layers': 4, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	55.60372635468333	11	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
2	18.261534441826868	31	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6"
3	33.02371793418824	21	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 5"
3	27.98216470016777	22	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	21.942785396983005	32	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6"
3	27.94334424780393	20	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5"
3	22.640785226546544	17	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	59.40952556261289	13	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	23.348158509526087	17	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5"
3	20.02595984494375	22	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	18.14765947498897	22	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	92.53151414978988	19	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	29.923530684764827	15	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5"
3	23.610465638622543	21	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6"
3	23.955525091337275	16	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6"
3	22.865239837509712	16	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 5, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	23.42795344962444	20	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6"
3	27.66343203219765	11	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	62.58096346006519	11	"'embed˙size': 256, 'hidden˙size': 256, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	21.897947927659008	28	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6"
3	31.92845911824304	13	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	19.3190541352914	14	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6, 'encoder˙attn': True, 'encoder˙append˙window': True"
3	24.751201634721	16	"'embed˙size': 512, 'hidden˙size': 512, 'num˙layers': 6"

Table 6: MCTS Hyperparameter Sweep Results with Interaction Terms (continued)