2022 Fourth International Conference on Transdisciplinary AI (TransAI) | 978-1-6654-7184-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/TransAI54797.2022.00012

A Hybrid Approach for Automatic Feedback Generation in Natural Language Programming

Yue Zhan and Michael S. Hsiao
Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA 24060, USA
{zyue91, hsiao}@vt.edu

***Abstract*—Developing a natural language programming (NLPr) tool is challenging due to the complex nature of natural language (NL). NLPr can be brittle and prone to failure when faced with unknown vocabulary or new sentence structures not supported by the system. In the absence of helpful feedback, the time it takes to complete the programming task might increase, and users might be discouraged. Recognizing the necessity of clear and actionable feedback, we propose a novel hybrid automatic feedback generation system in which a formal rule-based method is combined with a data-based multi-label classification (MLC) method to predict the user's intent and present examples of known-good ways to accomplish the intended or similar tasks. We evaluate the feedback system using two phases of end-user studies, demonstrating its usefulness in helping users troubleshoot their inputs and complete their tasks.**

***Index Terms*—Natural language programming, natural language processing, language validation, feedback generation.**

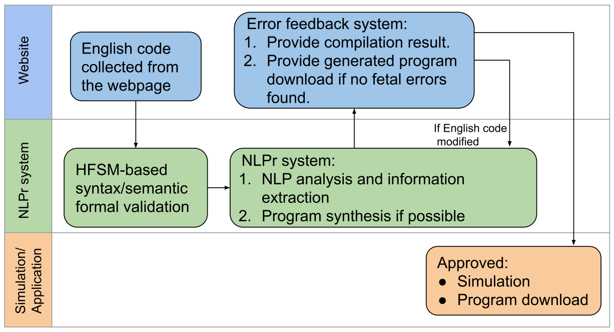

Website
English code collected from the webpage
Error feedback system:
1. Provide compilation result.
2. Provide generated program download if no fetal errors found.
If English code modified
NLPr system
HFSM-based syntax/semantic formal validation
NLPr system:
1. NLP analysis and information extraction
2. Program synthesis if possible
Simulation/ Application
Approved:
• Simulation
• Program download

Fig. 1: LEGO NLPr System Architecture

I. Introduction

Natural language programming (NLPr), a research field that aims to make computer programming more human-centric and enable users without prior coding experience to program computers with natural language, has been an active area of research in recent years as demand for no-code and low-code tools has grown both within the business and education communities. Research in NLPr is challenging as there is a considerable gap between natural language (NL) and programming language (PL). The former is expressive and natural, while the latter is precise and unambiguous [1]. Furthermore, designing an NLPr system is challenging due to the enormous number of language features. Thus, the language features that can be allowed in an NLPr system may only be a domain-specific subset of NL. For a given domain-specific NLPr application, a number of language features are selected to form a loosely controlled language subset analogous to a controlled natural language (CNL). Even so, the size of the language can still be very large or too restrictive. When the NLPr system does not accept an input sentence, it is crucial to provide helpful feedback and help the users achieve their intended goal. One way to minimize the impact of translation failures is to gracefully handle unexpected sentence structures and phrases that are not covered in the lexicon and grammar, recognizing the system's limits and communicating this information back to the user.

In order to maintain usability in the face of often murky input requirements and potential translation failures, it is essential that any experience with the system's limits results in a positive learning experience for the user, helping them build an understanding of the system that accurately approximates its capabilities and limits. Unfortunately, simply displaying alerts and generating generic error messages often fails to communicate the issue with users [2], sometimes because these messages lack the detailed information needed to understand the cause of an error [3]. Considering an error when programming with a conventional programming language, it is common for users to conduct additional research to understand the errors generated, either by reading dense and technical documentation or leveraging online resources like StackOverflow. As one of the goals of an NLPr system is to enable novice users to learn to code, such failures would be unacceptable. We also observe that users often ignore verbose and complex messages. These observations suggest that a human-centered approach is needed to help users understand why the system failed on their input and how to correct it.

In order to develop such a feedback engine, we build upon an interactive NLPr educational system for programming LEGO Mindstorms EV3 robots (LEGO robots) with English, called English code [4], [5], shown in Figure 1. The initial LEGO NLPr system uses a rule-based hierarchical finite state machine (HFSM) [6] as the formal validation step before feeding a user's English input to the NLPr system. Even though the context-sensitive HFSM-based validation can generate error messages for sentences that fail validation, it can only generate detailed error messages when it encounters vocabulary, sentence structure, and errors that are within expectation and have

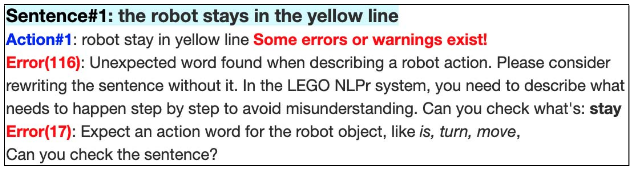

Sentence#1: the robot stays in the yellow line
Action#1: robot stay in yellow line **Some errors or warnings exist!**
Error(116): Unexpected word found when describing a robot action. Please consider rewriting the sentence without it. In the LEGO NLPr system, you need to describe what needs to happen step by step to avoid misunderstanding. Can you check what's: **stay**
Error(17): Expect an action word for the robot object, like *is, turn, move*,
Can you check the sentence?

Fig. 2: Low Quality Error Message Example

been written into the HFSM. When the HFSM encounters an unknown structure, vocabulary, or typo, it can only determine that the input does not match any known-good patterns, but it may be unable to describe precisely why the input cannot be accepted. As an example of this behavior, consider the erroneous sentence "*The robot stays in the yellow line.*" Upon processing this sentence, the HFSM-based validator determines that the word *stay* in the input is not in the known-good vocabulary, and it generates the error message shown in Figure 2. This error message indicates the word causing the problem, but it fails to provide the user with helpful information as the system was not built to generate detailed, quality error messages for this unforeseen input.Developers can patch up the HFSM when new structures/vocabularies are captured with manual effort. However, such an approach will require long-term maintenance. In the situation where the HFSM can only generate a generic error message, it would be helpful if additional feedback were provided to the user before the developer updates the HFSM, whether by giving insight as to why a more detailed error could not be generated, explaining to the user the system's understanding of what the user's input means, or providing suggestions on how to turn the erroneous input into one that passes validation.

Rather than relying solely on a rule-based error reporting system [6], which is only capable of generating helpful feedback for particular error classes built into the HFSM, we propose a hybrid approach that combines the existing rule-based error messages with suggestions generated using a data-based method derived from a multi-label classification (MLC) model. The hybrid feedback system is capable of handling both known and novel error patterns, leveraging the rule-based system to generate detailed error messages for known error patterns, and using the data-based method, trained on a corpus of past user inputs, to predict a user's intent and display recommendations for known-good approaches for the predicted intent. The data-based method accomplishes this by recommending sentences to users that may serve as templates for the task they want to accomplish. Incorporating the MLC for prediction means that these sentences can be more relevant to a user's desired task than the general error messages and sample sentences from the rule-based system, which may entirely miss the user's intent. The main challenge lies in training a reliable and trustworthy MLC model with the small dataset collected. A simple structured neural network is used, and the classification task is formed as a ranking problem to filter the prediction results with a confidence threshold to raise the quality of the suggestion(s) given to the user. We demonstrate the benefits of the proposed hybrid approach using two phases of end-user studies, observing more positive feedback for the hybrid system than the rule-based only system. In addition, we also observe that with the help of the debugging suggestions provided by the hybrid approach, it takes students less time and effort to use the LEGO NLPr platform to accomplish their given programming tasks.

II. Related Work

Deep learning neural network models are well known for the high accuracy they can achieve on multi-label classification tasks, and they have been demonstrated to be useful for tasks such as sentiment analysis [7]–[9] and topics labeling [10]–[12]. However, despite the well-established utility of neural networks for some NLP tasks, using neural network-based approaches for natural language programming and code synthesis is only a relatively new development. In work [13], the authors propose a *Tellina*, a system that translates NL to shell scripts using a recurrent neural network encoder-decoder approach. In work [14], an attention-based architecture termed *Latent Attention* is used to translate NL textual descriptions to If-Then programs for the IFTTT.com platform. *Deepcoder* [15] is a system that addresses inductive program synthesis tasks by training an RNN model with numeric input/output pairs and designing a domain-specific language based on SQL. In the work [16], the authors develop a syntactic Neural Model that parses NL descriptions into the general-purpose programming language Python. The above works pass their inputs directly to the neural translation component, without pre-processing or post-processing, to identify or mitigate mistranslations. This poses an issue for specific safety-critical tasks, such as real-time code synthesis and robot navigation, where it may be preferable for the generated code to do nothing at all rather than potentially put the robot at risk.

Within our domain of educational NLPr, it is crucial for an NLPr system to not only generate executable programs but for it also serves as an educational tool that enables users to learn how to create programs using English. Therefore, we focus on generating accurate predictions of what robot functionality a user intends to use in a command marked as erroneous. These predictions provide transparency into how the system interprets its inputs and aid in generating relevant recommendations for correcting common errors in English inputs. The main challenge of NLPr and traditional NLP tasks are semantic extraction, the process of parsing NL sentences to intermediate representations that can be further analyzed [17], [18]. Many works have been conducted in developing formal grammars for language representations [19]–[21]. There are works done using ML-based learning approaches, like semantic role labeling [22] and representation learning with graph convolution network (GCN) [23] and with Recurrent convolutional neural networks (RCNN) [24]. In our study, the function label semantic representation serves two purposes: representing the functions needed for program synthesis and the input sentences' casual spatial relationship.

III. Research Questions and Approach

A. The LEGO NLPr System

As explained above, there are many challenges when developing an NLPr system. First, the sentences may be ambiguous. For example, it can be difficult for a machine to process the sentence "*If the robot is blocked, it stops.*" Without prior direction from a user, the system does not have sufficient information to determine what would put the robot in the 'blocked' state. In the LEGO NLPr system, a domain-specific lexicon and function library $\mathcal{F}$ are utilized to restrict the language into a controlled natural language. Additionally, the sentence style is described as an object-oriented fashion to help disambiguate the NL semantics and reduce the space of ambiguity in the sentences. Using a controlled object-oriented language (COOL) is intended to ensure that users are writing unambiguous robot instructions and make the information extraction process an easier task. A COOL language model L is represented as $L = (O, A, P, R)$, where O represents the objects, and A represents the actions. The properties, P, are the properties affiliated with objects or actions. Finally, R stands for the hard-coded rules that apply to the system. The subset of natural language vocabulary and phrases relevant to LEGO robots is substantially smaller than the full scope of NL, the flexibility of the natural language subset makes the task of exhaustively building a rule-based system that can handle every possible sentence structure, vocabulary, and corner case difficult. Given the likelihood that a user may give the NLPr system an input the existing system did not cover, such failures must be identified and handled accordingly. In order to function effectively as a learning tool and minimize user frustration, the error detection system can be paired with an error reporting system that provides debugging information and suggestions. The COOL language model can be used as the skeleton to build an error-reporting feedback system. To understand how we might build an enhanced error reporting system for the LEGO NLPr system, we should first examine how the current error reporting and formal validation systems function.

B. Formal Validation And Error Reporting

In the work that utilized the HFSM system for validation, the authors noted that mistranslation could occur in situations where the translation component of the NLPr system generated a program even though the input was ambiguous [6]. For example, the true intentions of the sentence "*The robot does not want to go forward.*" might not even be apparent to a human. A formal validation can be applied to intuitively examine the input to catch any unexpected language components or potential errors in the sentence. The error message generated by the formal validation for each erroneous sentence is a program-centric error description explaining which part of the sentence might cause the problem companies with an integer number indicating the index in the full recognized error list, along with standardized example sentences. However, we found that it took some time for the users to process the program-centric error descriptions. When the error message listed multiple causes, the users would often only look at the first suggestion, subsequently relying on trial and error until their input was accepted. We theorize two possible reasons why error messages often confuse users:

- Case 1: The information provided is too general. For example, consider the sentence, "*The robot moves around.*" The formal validation system recognizes an issue and displays Error#30, indicating an unknown word *'around'* in the sentence. However, the generated error message does not provide strong guidance on fixing the sentence because there is not enough information about how the unknown word should be handled.

  Error(30): Unexpected word *word_token* is found and it might cause mistranslation. Please check the sentence for spelling or grammar errors.

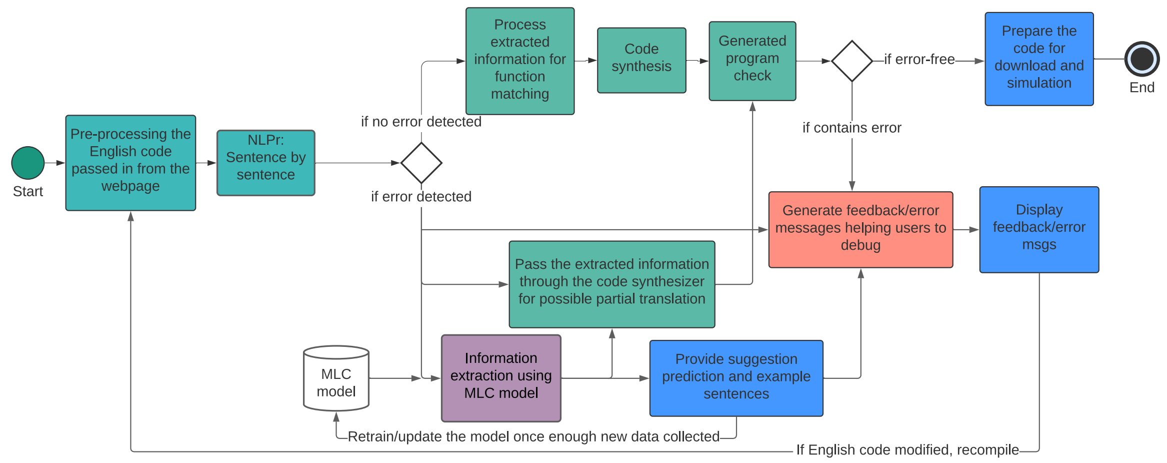
- Case 2: The suggestions provided in the error messages are hard-coded, with general examples, and there is no guarantee that they will be relevant to the task the user is trying to achieve. For example, if a conditional statement contains an ambiguous or unsupported condition, such as "if there is a color sensor at port 1, ...", the example sentences provided by Error #86 might not mention that conditional logic based on whether a sensor is connected to the rover is not supported.

  Error(86): Possibly ambiguous or unsupported conditional statement. You can ONLY check variable values or sensors in a conditional statement at the moment. For example, "If the ultrasound sensor sees anything within 5 inches,...", "If the color sensor sees black,...", "If the robot is happy,...", "If there is a black line, ...", "If var_1 is 5, ..."

The above examples show that the error messages should be more specific to the contexts of the sentences. Otherwise, it could confuse novice users about how to debug and fix their sentences. The trial and error debugging process that we observe, in combination with the fact that some error messages are not as helpful as expected, suggests a gap between the users' understanding of how the system processes sentences and how the NLPr system validates and translates them. Building a system that gives users insight into how the NLPr system understands input sentences could help bridge this gap, potentially reducing user confusion and speeding up debugging.

IV. Proposed Hybrid Feedback System

The approach we take here is to train a robust MLC ML model that extracts valuable information from inputs that fail validation and combine that information with the existing rule-based formal validation approach to provide richer debugging information. The workflow of the hybrid feedback system is shown in Figure 3. If the HFSM detects that the translation process would fail, the data-based ML model predicts the top-k ($k \leq 6$) function labels most likely to be associated with the input sentence. The final layer of the model uses



Start
Pre-processing the English code passed in from the webpage
NLPr: Sentence by sentence
if no error detected
if error detected
Process extracted information for function matching
Code synthesis
Generated program check
if error-free
if contains error
Prepare the code for download and simulation
End
Generate feedback/error messages helping users to debug
Display feedback/error msgs
Pass the extracted information through the code synthesizer for possible partial translation
MLC model
Information extraction using MLC model
Provide suggestion prediction and example sentences
Retrain/update the model once enough new data collected
If English code modified, recompile

Fig. 3: LEGO NLPr System Workflow

a sigmoid activation function where each label receives an independent prediction score. We then apply a threshold to the model's outputs, dropping all labels whose scores are below the threshold. We select a threshold of 1/41 in the actual implementation since there are a total of 41 classes, and we value maximizing recall over maximizing precision. We then take the top-k predictions over the threshold and use them to generate feedback for the user.

A. Learning Models and Dataset

There are many approaches for solving MLC tasks including *problem transformation* methods and *algorithm adaptation* methods [25]–[28]. The *problem transformation* methods transfer the multi-label learning problem into other well-established single-label learning problems, while the *algorithm adaptation* methods extend existing learning paradigms to handle multi-label data directly. Deep neural networks (DNN) are an example of ML algorithms that natively support MLC problems. In this work, we employ an NN-based *algorithm adaptation* method instead of using a *problem transformation* method [29]. The dataset used to train this model consists of English Codes collected from previous end-user studies, including both correct sentences that can be processed by the system and erroneous sentences that might not be translatable. Two phases of end-user studies were conducted with two groups of undergraduate students in the Spring and Fall semesters of 2021, respectively, and 16,378 total unique raw sentences were collected. The students were awarded extra credit for their courses as an incentive.

- Phase 1: The HFSM-based error reporting system.
- Phase 2: The hybrid feedback system.

Both phases include five different robot tasks. Sample robot tasks details are included in Appendix A. The difficulty of each task is slightly increased from one to the next to see how they ramp up with the new system. The number of participants and total submissions are shown in Table I.

	Phase 1		Phase 2	
	# of student	Total tries	# of student	Total tries
Task 1	90	2276	40	797
Task 2	63	4453	31	512
Task 3	60	2437	24	464
Task 4	60	1695	23	443
Task 5	61	1288	21	346

TABLE I: Total Submission Counts in End-user Study

The initial neural network model was trained with data collected from the Phase 1 user study and gradually updated with the data from Phase 2 when data became available. The collected data are fed through a pre-processing pipeline, which entails the replacement of abbreviations with full terms, a transformation from text to integer for numbers, lemmatization, and tokenization to clean up the noise and transform the data into a more digestible form for the downstream training process. 12,007 distinct sentences are collected in total after the pre-processing treatment, 10% is used for testing, and another 10% is used for validation. Each data sample is made of one sentence and its corresponding function labels, which represent the matching LEGO robot functions. A full list of the current function labels is shown in Table II. The labels for the correct sentences are auto-annotated by the system, while the erroneous sentences are manually labeled. The labels for each sentence consist of action labels, indicating robot actions needed, and condition labels, describing conditions that trigger actions if there are any. While each sentence may have a different number of labels, each sentence must have at least one action label, and conditional statements must have at least one valid condition label.

As mentioned above, only the top-k, with $k \leq 6$, results over the confidence threshold will be selected. This number 6 is chosen as no sentence in the dataset has more than six labels. After identifying the top-k most likely labels, sample sentences corresponding to these same labels are randomly selected from the training dataset and provided to the user as a

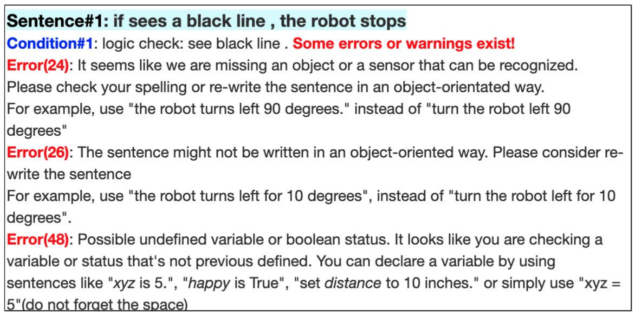
	Conditions
1	Use the ultrasound sensor in a conditional statement
2	Use two color sensors in a conditional statement
3	Use a color sensor in a conditional statement
4	Use a touch sensor in a conditional statement
5	Use a gyro sensor in a conditional statement
6	Use a Boolean variable in a conditional statement
7	Use a numerical variable in a conditional statement
8	Use a color variable in a conditional statement
	Actions
9	Set the power or speed of a motor
10	Modify the power or speed of a motor
11	This sentence might not be translatable
12	Break out of the main loop
13	Display a picture
14	Play a sound file
15	Display text
16	Read the color from a color sensor
17	Read the value from an ultrasound sensor
18	Read the value from a gyro sensor
19	Detect touch from a touch sensor
20	Time delay function
21	Line tracking function
22	Set a numeric variable
23	Set a Boolean variable
24	Pause the robot
25	Stop the robot/program forever
26	Turn the robot dir a certain number of degrees
27	Move the robot dir for a certain distance
28	Turn the robot dir for a particular length of time
29	Move the robot dir for a particular length of time
30	Turn the robot dir 90 degrees
31	Move the robot dir
32	Assign a sensor to a sensor port
33	Assign a motor to a motor port
34	Turn on an LED with a particular color
35	Turn off an LED
36	Pause the motor
37	Stop the motor forever
38	Move the motor dir for a certain degree/rotation
39	Move the motor dir for a particular length of time
40	Move the motor dir
41	Math arithmetic

TABLE II: LEGO NLPr function label list (41 classes in total)

reference for correctly accomplishing the task described by the label. The combination of rule-based and data-based methods allows the system to generate more helpful feedback based on the syntax and semantic information extracted from the user's input sentence. A user-friendly interface that explains the functions that might be needed for the sentences with focus and provides semantically relevant examples also encourages the users to fix the sentence instead of simply glancing over the error messages. Overall, this results in more relevant suggestions tailored to the task users are predicted to be interested in, helping them in their debugging process.

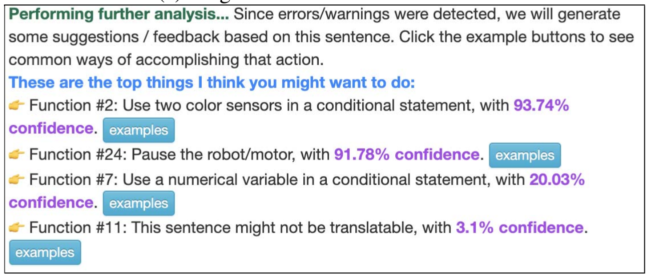
B. Full Pipeline Example

Take the erroneous sentence "*If sees a black line, the robot stops.*" as an example. Once passed into the LEGO NLPr system, this sentence will first be processed by the HFSM-based validator, which recognizes that either there is a missing object (Error#24) or the sentence is not written in an object-orientated manner (Error#26), or there are undefined variables



Sentence#1: if sees a black line , the robot stops
Condition#1: logic check: see black line . **Some errors or warnings exist!**
Error(24): It seems like we are missing an object or a sensor that can be recognized. Please check your spelling or re-write the sentence in an object-orientated way.
For example, use "the robot turns left 90 degrees." instead of "turn the robot left 90 degrees"
Error(26): The sentence might not be written in an object-oriented way. Please consider re-write the sentence
For example, use "the robot turns left for 10 degrees", instead of "turn the robot left for 10 degrees".
Error(48): Possible undefined variable or boolean status. It looks like you are checking a variable or status that's not previous defined. You can declare a variable by using sentences like "*xyz* is 5.", "*happy* is True", "set *distance* to 10 inches." or simply use "xyz = 5"(do not forget the space)

(a) Original HFSM-based feedback



Performing further analysis... Since errors/warnings were detected, we will generate some suggestions / feedback based on this sentence. Click the example buttons to see common ways of accomplishing that action.
These are the top things I think you might want to do:
👉 Function #2: Use two color sensors in a conditional statement, with **93.74% confidence.** examples
👉 Function #24: Pause the robot/motor, with **91.78% confidence.** examples
👉 Function #7: Use a numerical variable in a conditional statement, with **20.03% confidence.** examples
👉 Function #11: This sentence might not be translatable, with **3.1% confidence**. examples

(b) Proposed MLC top-k prediction results

Fig. 4: Error messages for example sentence

(Error#48) and then generate the error messages shown in Figure 4a. However, these error messages are generalized based on the sentence structure, and not clear what is missing from the sentence. The data-based method can help determine the underlying information implied by the sentence. It predicts that the sentence likely to refer to the following functions[1], as shown in Figure 4b:

- Function #2: a condition that requires color sensor function
- Function #24: robot/motor pause function
- Function #7: a condition that checks numeric variable

In particular, we are 93.74% confident that the user's conditional statement requires color sensors and provides some example sentences that require the same function:

> For the function label #2, these are some examples from the database:
> - If the color sensors see the black line, the robot turns right 20 degrees.
> - When the color sensors see green, the robot goes forward 20 cm, and the program ends.

Such a confidence probability score is returned from the sigmoid output layer with the binary cross-entropy loss function. By providing the above sample sentences as feedback, the hybrid feedback system can help the users identify the problem in the sentence, then look at the possible functions and fix the sentence. In this particular example, we just need to add the object *color sensor* in the condition:

[1]Function #11 is ignored. It refers to sentences that cannot be handled by the system and thus are treated as comments.

If the color sensor sees a black line, the robot stops.

By considering the actual user sentences provided by the suggestion engine, the user can better pinpoint and fix the error. More examples of cases demonstrating how the system helps the user write their sentence correctly and discuss are provided in the Experimental Results Section V-C.

V. Experimental Results

A. Model Evaluation Metrics

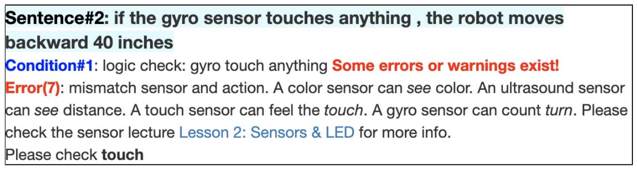
Evaluating MLC classification performance is more complex than evaluating binary classification since each prediction is no longer a simple "right or wrong". In MLC problems, it is possible for predictions to be partially correct, with some more correct than others. In this study, each sentence can map to a different number of labels. Since the top-k labels over the threshold are treated as the model's prediction result, we may predict extra false-positive labels because k is more significant than the number of ground truth labels for a particular sentence. Therefore, in addition to exact match accuracy, we also use macro and micro precision, recall, and F1 scores to evaluate the models' performance. The best threshold is chosen based on the highest exact match accuracy to compare the models fairly. We also report the label ranking average precision (LRAP) score, a commonly used metric in ranking problems that measure whether true labels are assigned higher scores than false labels, to describe model performance when the problem is formulated as a thresholdless ranking task.

B. Performance of the MLC model

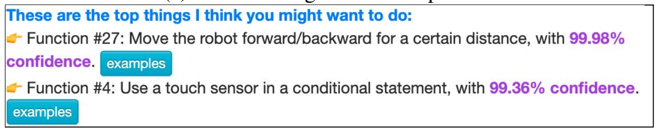
Due to the small size of the dataset and the class imbalance due to some functions being used less than others, the models chosen have relatively simple structures to prevent over-fitting. We developed ML models listed below and compare their performance in Table III:

- **LSTM**: Two bidirectional LSTM layers and two fully connected layers.
- **CNN**: One 1D convolution layer, a global max-pooling layer, and two fully connected layers.
- **LSTM+LSTM-pos**: A second bidirectional LSTM layer's outputs are concatenated and go through the fully connected layers.
- **CNN+LSTM-pos**: Use 1D CNN to learn the token streams and use LSTM to learn the POS information.
- **CNN+CNN-pos**: Use 1D CNN to learn the token streams and use CNN to learn the POS information.

The CNN-based model with additional POS input through CNN layers generally outperforms others, even though the performance is close across the models we developed, which might indicate the problem does not require a complex model considering the limited dataset size. The best threshold is chosen based on the threshold that achieves the best exact match accuracy. The micro precision and recall indicate that the models performed well across dominating classes, referring to the functions frequently used by the users. Especially,


Sentence#2: if the gyro sensor touches anything , the robot moves backward 40 inches
Condition#1: logic check: gyro touch anything **Some errors or warnings exist!**
Error(7): mismatch sensor and action. A color sensor can *see* color. An ultrasound sensor can *see* distance. A touch sensor can feel the *touch*. A gyro sensor can count *turn*. Please check the sensor lecture Lesson 2: Sensors & LED for more info.
Please check **touch**

(a) Error messages for Example 1


These are the top things I think you might want to do:
👉 Function #27: Move the robot forward/backward for a certain distance, with **99.98% confidence**. examples
👉 Function #4: Use a touch sensor in a conditional statement, with **99.36% confidence**. examples

(b) Function label predictions for Example 1

Fig. 5: Messages generated for Example 1

the CNN+CNN-pos model has an exact match accuracy of 76.16%, a micro precision rate of 96.03%, and a recall rate of 97.20% under the threshold of 0.32, indicating the true positive quality and quantity, thus proving the model's credibility and trustworthiness. The model achieves an LRAP score of 0.9827, which means the model is doing an adequately good job of correctly ranking the true labels at the top. The lower macro performances compared to micro average align with our known imbalanced data issue where some function labels only appear a few times. This suggests that future optimization includes auto-generating more sample sentences for the rare classes, and model fine-tuning is needed.

C. Case Studies and Discussion

Some specific user cases will be discussed in this section to illustrate how the feedback aids the users in debugging their English code.

Example 1: If the gyro sensor touches anything, the robot moves backward 40 inches.

In Example 1, Error#7: mismatch sensor and action, is reported, and some basic information related to the error is provided, as shown in Figure 5a. While it is not clear whether the user misused a sensor or action verb, function prediction shown in Figure 5b strongly suggests that this sentence indicates the touch sensor function.

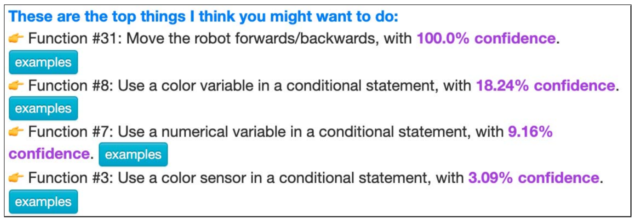
While the suggested labels produced by the MLC system often cover a user's intent, this is not always the case. The MLC model mispredicts more often in rare functions with less representation in the dataset or was caused by misspellings. An example of a situation where the prediction system makes an incorrect prediction is given below. However, the hybrid nature of our approach still generates a useful rule-based message that helps a user debug their input.

Example 2: The robot moves forward until it collides with the wall.

The functions predicted by the MLC model for this input are shown in Figure 6a. This is an interesting example of MLC mispredicting the needed sensor. While a human may note that the user likely wants to use a touch sensor, the MLC model incorrectly predicts that a color sensor might be needed. One possible reason for the misprediction could be that the

	Threshold	Exact match	Micro precision	Micro recall	Micro F1	Macro precision	Macro recall	Macro F1	LRAP
LSTM	0.35	0.7428	0.9553	0.9598	0.9576	0.6693	0.6557	0.6624	0.9728
LSTM+LSTM-pos	0.43	0.7605	0.9664	0.9582	0.9623	0.6649	0.6413	0.6529	0.9762
CNN	0.29	0.7306	0.9600	0.9519	0.9559	0.6724	0.6068	0.6379	0.9722
CNN+LSTM-pos	0.55	0.7428	**0.9729**	0.9482	0.9604	**0.7153**	0.6289	0.6693	0.9782
CNN+CNN-pos	0.32	**0.7616**	0.9603	**0.9720**	**0.9661**	0.6780	**0.6733**	**0.6756**	**0.9827**

TABLE III: Comparisons of NN-based models

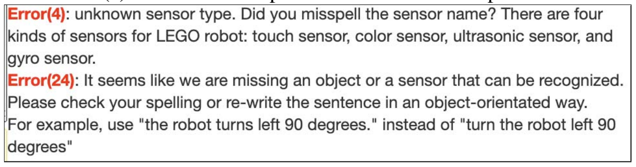

These are the top things I think you might want to do:
👉 Function #31: Move the robot forwards/backwards, with **100.0% confidence**.

👉 Function #8: Use a color variable in a conditional statement, with **18.24% confidence**.

👉 Function #7: Use a numerical variable in a conditional statement, with **9.16% confidence**.

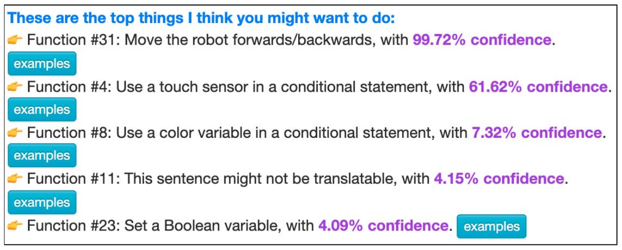
👉 Function #3: Use a color sensor in a conditional statement, with **3.09% confidence**.

(a) Function label predictions #1 for Example 2


Error(4): unknown sensor type. Did you misspell the sensor name? There are four kinds of sensors for LEGO robot: touch sensor, color sensor, ultrasonic sensor, and gyro sensor.
Error(24): It seems like we are missing an object or a sensor that can be recognized. Please check your spelling or re-write the sentence in an object-orientated way.
For example, use "the robot turns left 90 degrees." instead of "turn the robot left 90 degrees"

(b) Error messages for Example 2

Fig. 6: Messages generated for Example 2


These are the top things I think you might want to do:
👉 Function #31: Move the robot forwards/backwards, with **99.72% confidence**.

👉 Function #4: Use a touch sensor in a conditional statement, with **61.62% confidence**.

👉 Function #8: Use a color variable in a conditional statement, with **7.32% confidence**.

👉 Function #11: This sentence might not be translatable, with **4.15% confidence**.

👉 Function #23: Set a Boolean variable, with **4.09% confidence**.

Fig. 7: Function label predictions #2 for Example 2

verb *collides* and the touch sensor function are not frequently used together in the training dataset, and that conditional move instructions most often deal with the color sensor for sensing lines on the ground/floor. However, even though the MLC system fails here, the hybrid system is still able to help the user as the rule-based error reporting system generates the message shown in Figure 6b, correctly noting that the user did not explicitly name a sensor in their input and telling the user which sensors are available. After receiving the feedback from the hybrid system, the user corrected the sentence to version 2 " *The robot moves forward until it touches the wall.*" which, while still erroneous, is successfully handled by the MLC system. It identifies that a touch sensor is in use, as shown in Figure 7. At this point, the error messages and function label suggestions guide the user to the third version of the sentence "*The robot moves forward until the touch sensor is pressed.*" Cases like this suggest that expanding the dataset during the training may allow the system to be of greater utility.

VI. Conclusion

This paper presented a hybrid approach for generating meaningful feedback to help users program a LEGO robot using natural language. The hybrid system combines information from a rule-based HFSM validation system and a data-based machine learning approach and provides messages and examples to users using a user-centered human-computer interface. Our analysis of the study results shows how the addition of intent-aware recommendations helps users convert untranslatable sentences into ones that the NLPr system can successfully translate into a corresponding executable program. The proposed hybrid approach shows promising performance for the educational LEGO NLPr system, and a similar approach could be adapted for use with a broad range of NLPr assistant toolkits.

There are a few ways this work could be expanded in the future. One particularly intriguing possibility for improving the robustness of the system would be to adopt continual learning, which allows the system to adapt iterative feedback and learn new terms, structures, and tasks in an online manner. Continual learning also keeps the model from being stale and allows users to 'teach' the system to understand how they use English.

Acknowledgment

This research is supported in part by NSF grant 2101021.

References

[1] T. Kuhn, "A survey and classification of controlled natural languages," *Comput. Linguist.*, vol. 40, no. 1, pp. 121–170, Mar. 2014.

[2] G. Marceau, K. Fisler, and S. Krishnamurthi, "Mind your language: On novices' interactions with error messages," in *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, ser. Onward! 2011. New York, NY, USA: Association for Computing Machinery, 2011, p. 3–18. [Online]. Available: https://doi.org/10.1145/2048237.2048241

[3] H. Kadekar, S. Sohoni, and S. Craig, "Effects of error messages on students' ability to understand and fix programming errors," in *Frontiers in Education*, ser. Proceedings - Frontiers in Education Conference, FIE. Institute of Electrical and Electronics Engineers Inc., Mar. 2019, 48th Frontiers in Education Conference, FIE 2018 ; Conference date: 03-10-2018 Through 06-10-2018.

[4] Y. Zhan and M. S. Hsiao., "A natural language programming application for lego mindstorms EV3," in *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality, AIVR 2018, Taichung, Taiwan, December 10-12, 2018.* IEEE Computer Society, 2018, pp. 27–34. [Online]. Available: https://doi.org/10.1109/AIVR.2018.00012

[5] Y. Zhan and M. S. Hsiao, "Breaking down high-level robot path-finding abstractions in natural language programming," in *Proceedings of the 4th Workshop on Natural Language for Artificial Intelligence (NL4AI 2020) co-located with the 19th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2020), Anywhere, November 25th-27th, 2020*, ser. CEUR Workshop Proceedings, vol. 2735. CEUR-WS.org, 2020, pp. 59–72. [Online]. Available: http://ceur-ws.org/Vol-2735/paper33.pdf

[6] Y. Zhan and M. Hsiao, "Formal validation for natural language programming using hierarchical finite state automata," in *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 1: NLPinAI,*, INSTICC. SciTePress, 2021, pp. 506–515.

[7] L. Dong, F. Wei, C. Tan, D. Tang, M. Zhou, and K. Xu, "Adaptive recursive neural network for target-dependent Twitter sentiment classification," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, Jun. 2014, pp. 49–54. [Online]. Available: https://www.aclweb.org/anthology/P14-2009

[8] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014. [Online]. Available: http://arxiv.org/abs/1408.5882

[9] 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*. Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 1422–1432. [Online]. Available: https://www.aclweb.org/anthology/D15-1167

[10] N. Aletras and A. Mittal, "Labeling topics with images using a neural network," in *Advances in Information Retrieval*, J. M. Jose, C. Hauff, I. S. Altıngovde, D. Song, D. Albakour, S. Watt, and J. Tait, Eds. Cham: Springer International Publishing, 2017, pp. 500–505.

[11] C. Zhou, C. Sun, Z. Liu, and F. C. M. Lau, "A C-LSTM neural network for text classification," *CoRR*, vol. abs/1511.08630, 2015. [Online]. Available: http://arxiv.org/abs/1511.08630

[12] F. Monti, F. Frasca, D. Eynard, D. Mannion, and M. M. Bronstein, "Fake news detection on social media using geometric deep learning," *CoRR*, vol. abs/1902.06673, 2019. [Online]. Available: http://arxiv.org/abs/1902.06673

[13] X. V. Lin, "Program synthesis from natural language using recurrent neural networks," 2017.

[14] C. Liu, X. Chen, E. C. Shin, M. Chen, and D. Song, "Latent attention for if-then program synthesis," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4574–4582. [Online]. Available: http://papers.nips.cc/paper/6284-latent-attention-for-if-then-program-synthesis.pdf

[15] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, "Deepcoder: Learning to write programs," *CoRR*, vol. abs/1611.01989, 2016. [Online]. Available: http://arxiv.org/abs/1611.01989

[16] P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," *CoRR*, vol. abs/1704.01696, 2017. [Online]. Available: http://arxiv.org/abs/1704.01696

[17] Y. Artzi and L. Zettlemoyer, "Bootstrapping semantic parsers from conversations," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '11. USA: Association for Computational Linguistics, 2011, p. 421–432.

[18] L. Dong and M. Lapata, "Language to logical form with neural attention," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 33–43. [Online]. Available: https://www.aclweb.org/anthology/P16-1004

[19] L. S. Zettlemoyer and M. Collins, "Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars," *CoRR*, vol. abs/1207.1420, 2012. [Online]. Available: http://arxiv.org/abs/1207.1420

[20] J. Berant, A. Chou, R. Frostig, and P. Liang, "Semantic parsing on Freebase from question-answer pairs," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1533–1544. [Online]. Available: https://www.aclweb.org/anthology/D13-1160

[21] V. Raman, C. Lignos, C. Finucane, K. Lee, M. Marcus, and H. Kress-Gazit, "Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using natural language," in *Robotics: Science and Systems*, 2013.

[22] D. Marcheggiani and I. Titov, "Encoding sentences with graph convolutional networks for semantic role labeling," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 1506–1515. [Online]. Available: https://www.aclweb.org/anthology/D17-1159

[23] L. Zhang, H. Song, and H. Lu, "Graph node-feature convolution for representation learning," *CoRR*, vol. abs/1812.00086, 2018. [Online]. Available: http://arxiv.org/abs/1812.00086

[24] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. AAAI Press, 2015, p. 2267–2273.

[25] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown, "Learning multi-label scene classification," *Pattern Recognition*, vol. 37, no. 9, pp. 1757 – 1771, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320304001074

[26] Min-Ling Zhang and Zhi-Hua Zhou, "A k-nearest neighbor based algorithm for multi-label classification," in *2005 IEEE International Conference on Granular Computing*, vol. 2, 2005, pp. 718–721 Vol. 2.

[27] N. Spolaôr, E. A. Cherman, M. C. Monard, and H. D. Lee, "A comparison of multi-label feature selection methods using the problem transformation approach," *Electronic Notes in Theoretical Computer Science*, vol. 292, pp. 135–151, 2013, proceedings of the XXXVIII Latin American Conference in Informatics (CLEI). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1571066113000121

[28] A. Clare and R. D. King, "Knowledge discovery in multi-label phenotype data," in *Principles of Data Mining and Knowledge Discovery*, L. De Raedt and A. Siebes, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 42–53.

[29] Y. Zhan and M. S. Hsiao., "Multi-label classification on natural language sentences for video game design," in *2019 IEEE International Conference on Humanized Computing and Communication (HCC)*, 2019, pp. 52–59.

Appendix A
Example Robot Task

Phase 2 Exercise 3 Task Description: Write English code to fulfill the task demonstrated in Figure 8:

1) From point 1, moves to the obstacle at point 2.
2) Detect the obstacle with any sensor.
3) Back up for a bit and stop.

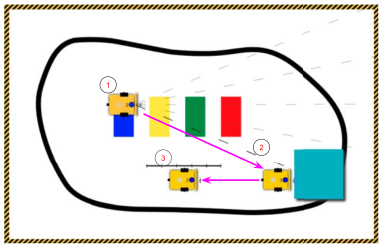


Fig. 8: Phase 2 Exercise 3 Task Requirement

Example English code #1 from user study:

> The robot turns right 90 degrees. The robot moves forward 50 cm. The robot turns left 90 degrees. The robot moves forward 60 inches. There is a touch sensor at port S1. If the touch sensor is pressed, the robot moves backward 20 inches. Program ends.

Example English code #2 from user study:

> Read distance from the ultrasound sensor. The robot turns right 90 degrees. The robot moves forward 50 cm. The robot turns left 90 degrees. The robot moves forward 50 inches. While the distance is more than 2 cm, the robot moves forward. The robot moves backward 20 inches, and the program ends.