UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO - UFERSA
PRÓ-REITORIA DE GRADUAÇÃO
CENTRO MULTIDISCIPLINAR DE PAU DOS FERROS
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO - BTI

ÂNGELO GABRIEL LOPES DA SILVA

UMA FERRAMENTA PARA CORREÇÃO DE ERROS SINTÁTICOS USANDO APRENDIZADO DE MÁQUINA PARA PROGRAMAÇÃO INTRODUTÓRIA

PAU DOS FERROS
2020

ÂNGELO GABRIEL LOPES DA SILVA

UMA FERRAMENTA PARA CORREÇÃO DE ERROS SINTÁTICOS USANDO APRENDIZADO DE MÁQUINA PARA PROGRAMAÇÃO INTRODUTÓRIA

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Tecnologia da Informação.

Orientador: Prof. Dr. Reudismam Rolim de Sousa

PAU DOS FERROS
2020

S586f

Silva, Ângelo Gabriel Lopes da.
Uma Ferramenta para Correção de Erros Sintáticos Usando Aprendizado de Máquina para Programação Introdutória / Ângelo Gabriel Lopes da Silva. - 2020.
67 f. : il.

Orientador: Reudismam Rolim de Sousa.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Tecnologia da Informação, 2020.

1. Correção de código. 2. Erro sintático. 3. Aprendizado de máquina. I. Sousa, Reudismam Rolim de, orient. II. Título.

ÂNGELO GABRIEL LOPES DA SILVA

UMA FERRAMENTA PARA CORREÇÃO DE ERROS SINTÁTICOS USANDO APRENDIZADO DE MÁQUINA PARA PROGRAMAÇÃO INTRODUTÓRIA

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Tecnologia da Informação.

Defendida em: 05 / 02 / 2020 .

BANCA EXAMINADORA

Reudismam Rolim de Souza
Prof. Dr. Reudismam Rolim de Souza (UFERSA)
Presidente

Felipe Torres Leite
Prof. Me. Felipe Torres Leite (UFERSA)
Membro Examinador

Robson Locatelli Macedo
Prof. Me. Robson Locatelli Macedo (UFERSA)
Membro Examinador

AGRADECIMENTOS

Agradeço, primeiramente à Deus, pelo dom da vida e por ter me proporcionado chegar ao fim deste primeiro ciclo. A meus pais, Evanio Lopes da Silva e Adalmária Lopes e minha avó Maria das Dores Lopes, por todo incentivo, confiança, amor e carinho contribuindo diretamente para que eu pudesse chegar a esse momento.

A minha namorada Fernanda Kiarely por toda atenção, ajuda e apoio ofertado, seja nas horas boas ou ruins, pelos momentos de companheirismo, descontração, carinho e pela compreensão aos momentos de ausência, contribuindo assim para que pudesse chegar ao término deste ciclo.

A minha irmã Grazielle Lopes por todo apoio e companheirismo durante essa jornada percorrida, aos meus amigos e colegas de faculdade por todo aprendizado adquirido nesse percurso.

A meu orientador Reudismam Rolim por todo aprendizado, empenho, tempo ofertado, para que esse trabalho fosse concretizado e também pelos momentos de conversa e descontração.

RESUMO

Ao longo dos cursos relacionados à computação, os estudantes precisam resolver problemas de programação. Neste sentido, alguns estudantes podem apresentar dificuldades para corrigir problemas sintáticos contidos em o seu código. Apesar de algumas ferramentas como os ambientes de desenvolvimento integrado mostrarem os possíveis problemas, muitas vezes a solução exposta para resolvê-la, não ajuda o estudante na resolução dos mesmos, podendo a linha marcada como problemática se mostrar distante da linha fornecida pelo compilador. Neste trabalho é proposto uma ferramenta para correção de erros sintáticos usando aprendizado de máquina. Antes de partir para a construção da ferramenta em si, foi realizada uma revisão sistemática da literatura sobre as técnicas de aprendizagem de máquina que foram utilizadas para auxiliar estudantes na resolução de problemas de programação. Uma vez realizada a revisão sistemática da literatura, foi proposta uma ferramenta para correção automática de problemas sintáticos. A ferramenta utiliza uma rede neural artificial LSTM e recebe como entrada um programa incorreto e fornece sugestões de que *tokens* da linguagem que poderiam ser utilizados para resolver um problema sintático presente no código de um estudante.

Palavras-chave: Correção de código. Erro sintático. Aprendizado de máquina

ABSTRACT

Along computer-related courses, students need to solve programming assignments. In this sense, some students may have difficulties in correcting syntactic problems that their code presents. Despite the fact that some tools such as integrated development environments present possible problems, often the solution presented to solve the problem does not help the student in solving the problem, and the line marked as problematic may be distant from the line provided by the compiler. In this work, a tool to correct syntactic errors using machine learning was proposed. Before starting to build the tool itself, a systematic literature review was carried out on the machine learning techniques that were used to assist students in solving programming problems. Once the systematic literature review was carried out, a tool for automatic correction of syntactic problems was proposed. The tool uses an LSTM artificial neural network and receives an incorrect program as input and provides suggestions that language tokens could be used to solve a syntactic problem that a student's code presents.

Keywords: *Code correction. Syntactic error. Machine learning*.

LISTA DE ABREVIATURAS E SIGLAS

ACM	*Association for Computing Machinery*
AIED	*International Conference on Artificial Intelligence in Education*
ANTLR	*Another Tool for Language Recognition*
AST	Árvore Sintática Abstrata
CLOM	Cursos Livres Online Massivos
EASEAI	*Education through Advanced Software Engineering and Artificial Intelligence*
EBNF	*Extended Backus–Naur Form*
ESEC/FSE	*European Software Engineering Conference and Symposium on the Foundations of Software Engineering*
IA	Inteligência Artificial
ICAST	*International Conference on Advancements of Science and Technology*
ICER	*International Computing Education Research*
ICSE	*International Conference on Software Engineering*
ICSE-SEET	*International Conference on Software Engineering: Software Engineering Education and Training*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
IJET	*International Journal of Engineering and Technology*
ITiCSE	*Innovation and Technology in Computer Science Education*
ITS	*Intelligent Tutoring Systems*
MOOC	*Massive Open Online Course*
OO	Orientada a Objeto
OOPSLA	*Object-oriented Programming, Systems, Languages, and Applications*
PLDI	*Programming Language Design and Implementation*
RAP	Reparador Automático de Programas
RNAs	Redes Neurais Artificiais
RNN	*Recurrent Neural Network*
SBC	Sociedade Brasileira de Computação
SEEL	*Smart Education and E-Learning*
SoSyM	*International Journal on Software and Systems Modeling*

SPLASH	*Systems, Programming, Languages, and Applications: Software for Humanity*
UFERSA	Universidade Federal Rural do Semi-Árido

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE GRÁFICOS

SUMÁRIO

1. INTORDUÇÃO

Atualmente, os sistemas computacionais estão presentes nos mais diversos ambientes, desde sistemas críticos, tais como controladores de aeronaves, até softwares de prateleira usados no dia-a-dia, tais como os processadores de texto (SOMMERVILLE, 2011). Isso possibilitou tanto o acesso à informação, quanto, o aumento da produtividade na realização de tarefas, seja no cotidiano pessoal, no âmbito empresarial ou acadêmico. Santos et al. (2018a) ressalta a importância da computação na sociedade como um instrumento de ampliação de possibilidades e de geração de novas ideias, o que impulsionou iniciativas em diversos países para introduzi-la como ciência para todos.

Para o desenvolvimento de sistemas de computação, além de outros conhecimentos voltados à engenharia de software, necessita-se do conhecimento de programação, que envolve desde desenvolver a lógica para construir o sistema até a aplicação de uma linguagem de programação. Neste sentido, precisa-se desenvolver estas competências, em especial, em diferentes estágios do ensino desde o ensino básico até o ensino superior (SOUSA, 2019). Por exemplo, Royal (2012) mostra que o ensino da computação é adotado nos componentes curriculares do ensino médio – educação básica – em países como Reino Unido, Nova Zelândia, Alemanha, Índia e Coreia do Sul. Desta maneira, destaca-se a importância da computação no crescimento intelectual, social e profissional dos estudantes e no desenvolvimento de cada país.

A iniciativa de implantação também ocorreu no Brasil. Em 2004, no XXIV Congresso da Sociedade Brasileira de Computação (SBC), foi dialogada a viabilidade da implantação da disciplina de programação no ensino médio (JÚNIOR. 2005), essa iniciativa foi aprovada a fim de desenvolver as competências dos alunos nessa área. Recentemente, foram desenvolvidos parâmetros para inserção da computação, não somente no ensino médio, mas ao longo da educação básica – ensino infantil, fundamental e médio (SBC. 2017).

Entretanto, tais parâmetros na maioria das vezes não são aplicados no cotidiano de várias instituições de ensino, sejam elas públicas ou privadas da educação básica. Gerando assim, dificuldades no aprendizado e na resolução de tarefas de disciplinas introdutórias de programação aos ingressantes no ensino superior em cursos na área de computação. Moreira et al. (2018) constatou que 87,27% dos alunos entrevistados, na Universidade Federal Rural do Semi-Árido (UFERSA), nunca utilizaram uma linguagem de programação antes de cursar uma

disciplina que requeresse tal conhecimento, e apenas 12,72% já haviam utilizado uma linguagem de programação anteriormente.

Ainda segundo Moreira et al. (2018), as principais dificuldades enfrentadas pelos alunos, referentes ao entendimento de programação são a dificuldade de compreender a lógica de programação e de entender a sintaxe da linguagem. De acordo Tabanao et al. (2008) até mesmo um texto simples pode tornar-se um obstáculo para alunos que ingressam no estudo de programação. Santos et al. (2018b) destaca que, além de conhecer a semântica de uma linguagem de programação específica, é crucial aprender a inserir símbolos particulares de cada linguagem na ordem correta para que o computador possa produzir a saída esperada. Tabanao et al. (2008) destaca que o posicionamento correto dos símbolos – característicos de cada linguagem de programação – de sintaxe, é um dos erros mais cometidos em programação e até mesmo por programadores experientes.

1.1.Problemática

No contexto educacional, diversos estudantes que adentram em instituições de ensino superior na área de computação, tais como Tecnologia da Informação, Engenharia de Software, Engenharia da Computação, Ciência da Computação, entre outros, manifestam dificuldades em disciplinas introdutórias de programação. De acordo com Moreira et al. (2018), os discentes costumam apresentar dificuldades na aprendizagem de conteúdos que envolvam habilidades de resolver tarefas de programação.

Segundo Gomes (2010), o ensino de programação objetiva despertar as habilidades dos alunos em desenvolver sistemas computacionais capazes de solucionar problemas reais. No entanto, os estudantes constam dificuldades para desenvolver tais habilidades, sendo que as principais dificuldades estão relacionadas à compreensão das noções básicas da linguagem, para resolver os problemas reais.

A dificuldade em assimilar as noções básicas da linguagem, principalmente no tocante a sintaxe da linguagem, é um dos principais problemas enfrentados pelos alunos durante seu aprendizado. Segundo Moreira et al. (2018), no tocante aos desafios de aprendizagem de programação introdutória, 34,54% dos estudantes entrevistados afirmaram encontrar maiores dificuldades na compreensão da sintaxe da linguagem. Dentre os conteúdos ministrados na

disciplina de algoritmos, as maiores dificuldades dos alunos foram os conteúdos de funções e estrutura de repetição, ambos compreendem um percentual de 45,45 %.

Para resolver as tarefas de programação, os estudantes precisam conhecer os elementos da linguagem e também inserir estes elementos na ordem correta. Quando inseridos na ordem incorreta ou em locais inapropriados, são gerados erros de compilação que podem ser difíceis de serem entendidos por estudantes novatos de programação. Apesar de os Ambientes de Desenvolvimento Integrado (IDEs - do inglês *Integrated Development Environment*) fornecerem explicações sobre o problema, muitas vezes, essas explicações não vão de encontro com o erro apresentado pelo código, podendo o local a ser modificado para resolver o erro se encontrar muito distante da linha apresentada pelo compilador/IDE e a mensagem não ter qualquer relação com o problema a ser resolvido (Santos et al., 2018b).

Segundo Brown (2014), os erros mais corriqueiros entre os estudantes de programação estão relacionados a chaves não balanceadas. Os compiladores não dispõem de ajuda eficiente para constatação do local de tais erros (SANTOS et al., 2018b), além de emitirem mensagem de relatórios de erros complexas para muitos estudantes que não possuem muita prática com a programação, o que acaba não se tornando útil para resolução da tarefa do estudante.

Segundo Becker (2015), as referidas mensagens podem tornar-se um obstáculo para o avanço do discente e ser uma possível fonte de frustração. Gerando assim o desestímulo do aluno, quando não conseguem solucionar atividades requeridas pelos professores das disciplinas de programação mediante a erros de sintaxe e por não conseguir decifrar a mensagem de erro relatada pelo compilador, o que ocasiona uma elevação nos índices de reprovação, desistência da disciplina ou a evasão do aluno do curso ou da instituição de ensino.

1.2. Justificativa

Neste trabalho, será utilizada Inteligência Artificial (IA), mais precisamente, a subárea *machine learning* (aprendizado de máquina), com técnicas de processamento de linguagem natural, para criar uma ferramenta que possam auxiliar alunos que estão respondendo tarefas de programação a localizar e solucionar seus respectivos erros de sintaxe. Porém, é importante destacar que existem diferentes subáreas da IA e diferentes metodologias, passíveis de desenvolvimento voltadas para a melhoria educacional do aluno. Por exemplo Santos (2018b)

apresenta uma ferramenta denominada *Sensibility* desenvolvida utilizando uma arquitetura de redes neurais chamada de LSTM, com propósito de corrigir e encontrar erros de programação. Becker (2016) buscou desenvolver uma abordagem para aperfeiçoar as mensagens de erros relatadas pelos compiladores para prover melhor desempenho dos alunos.

Redes Neurais Recorrentes (RNNs) foram instruídas e aplicadas no código-fonte para proporcionar ao programador maior praticidade ao escrever o código fonte, independentemente da linguagem aplicada (WHITE et al., 2015; RAYCHEV et al., 2014). Hellendoorn (2017) argumentaram a efetividade da aprendizagem profunda – RNNs e LSTM – para fornecer sugestões de utilização da sintaxe da linguagem.

Considerando a necessidade dos estudantes novatos em programação de entender a sintaxe da linguagem de programação, o trabalho busca entender, evoluir e aplicar técnicas de inteligência artificial para identificar e sugerir correções de sintaxe com estudantes da UFERSA.

1.3. Objetivos

- **Objetivo Geral**

O objetivo desenvolver uma técnica que utilize processamento de linguagem natural, e que possibilite auxiliar os estudantes na localização e correções de erros de sintaxe, durante a realização de tarefas.

- **Objetivos Específicos**

Para alcançar o objetivo geral destacado, os seguintes objetivos específicos serão realizados:

a) Desenvolvimento de uma revisão sistemática da literatura, no qual trata-se as seguintes questões de pesquisa:

- RQ1 – Quais as técnicas empregadas para auxiliar estudantes na resolução de tarefas de programação, utilizando aprendizado de máquina (Processamento de Linguagem Natural)?

- RQ2 - Quais os métodos que possibilitam a identificação e correção de erros sintáticos?

b) Estudar o modelo de aprendizado de máquina *Long Short-Term Memory* (LSTM);

c) Desenvolver uma ferramenta computacional que utilize processamento de linguagem natural que localize e possibilite uma possível correção de erros de sintaxe.

1.4. Metodologia

De modo geral, para elaboração deste trabalho, primeiramente, realizou-se uma revisão sistemática da literatura para verificar as diversas contribuições existentes a respeito do tema proposto. Em seguida, efetiva-se estudos a respeito dos modelos de aprendizado de máquina LSTM, modelos estes, utilizados na última etapa do projeto que é constituída pela elaboração de uma ferramenta computacional que localiza e possibilita uma possível correção de erros de programação.

A revisão sistemática da literatura utilizou as bases de dados da ACM *Digital Library*, IEEE *Explore* e Scopus, com o propósito de averiguar os métodos e ferramentas existentes, e assim, analisar os aspectos positivos, negativos e limitações existentes entre as diversas ferramentas.

Por fim, foram realizados estudos relacionados aos modelos de aprendizado de máquina LSTM, que são adotados por diversos autores para constatação e localização de erros de sintaxe em diferentes linguagens de programação (CAMPBELL, 2014; BHATIA, 2016; SANTOS, 2018b), tornando-se assim crucial para o desenvolvimento de uma ferramenta computacional que utilize processamento de linguagem natural para localizar e possibilitar uma possível correção de erros de programação.

Este trabalho está organizado em 5 capítulos incluindo esta introdução. No Capítulo 2 é apresentada a fundamentação teórica; o Capítulo 3 destina-se a uma revisão sistemática sobre as técnicas que utilizam processamento de linguagem natural para auxiliar estudantes/programadores na resolução de tarefas de programação; no Capítulo 4 realiza-se a descrição da ferramenta proposta por este trabalho; e por fim, no Capítulo 5 são apresentadas as considerações finais e trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma visão geral dos conceitos fundamentais para compreensão deste trabalho, tanto conceitos básicos referentes ao aprendizado de máquina, mas especificamente redes neurais (Seção 2.1), quanto análise léxica e sintática (Seção 2.2), e a ferramenta *Another Tool For Language Recognition* (ANTLR) (Seção 2.3), que serão base para o modelo apresentado.

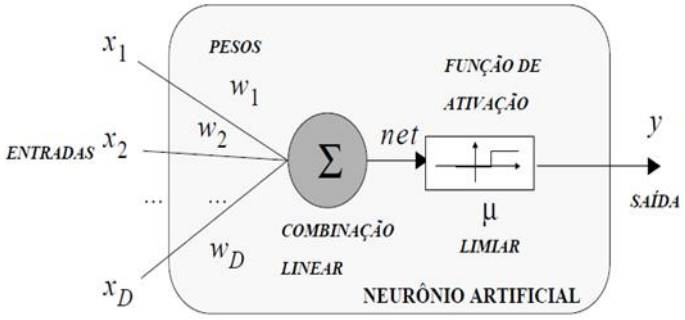
2.1. Redes Neurais Artificiais

Redes neurais artificiais (RNAs) são modelos paramétricos não-lineares, constituído por unidades de processamento baseadas em funções matemáticas (BRAGA et al., 1998; HAYKIN, 1999). As RNAs são baseadas no funcionamento de um cérebro, composta por unidades que são equivalentes a neurônios e funções de ativação, responsáveis por transmitir as informações entre os neurônios (NELSON, 2017).

A partir de dados passados de instâncias do domínio e os seus respectivos rótulos (dados de treino), as RNAs "aprendem" sobre esse conjunto de dados para classificá-los da forma mais adequada (GORGENS et al., 2009). Essas redes ainda são capazes de extrair características não implícitas das instâncias presentes no conjunto de treino (KOVACS, 1996).

Uma RNA possui como característica elementos como a arquitetura e o algoritmo de aprendizagem, tais características se constituem mediante o paradigma de como a rede é treinada. O algoritmo de aprendizagem processa os dados de treinamento e guarda o conhecimento dentro de parâmetros variáveis da rede denominados de pesos. Por outro lado, a arquitetura define como a rede é organizada, como por exemplo quantos neurônio a rede terá em suas camadas. A Figura 1 apresenta um modelo simplificado de uma RNA proposta por McCulloch e Pitts (1943) que busca replicar o funcionamento biológico que ocorrem dentro de uma célula do sistema nervoso (RAUBER, 2005).

Figura 1: Modelo de uma RNA



Fonte: RAUBER, 2005.

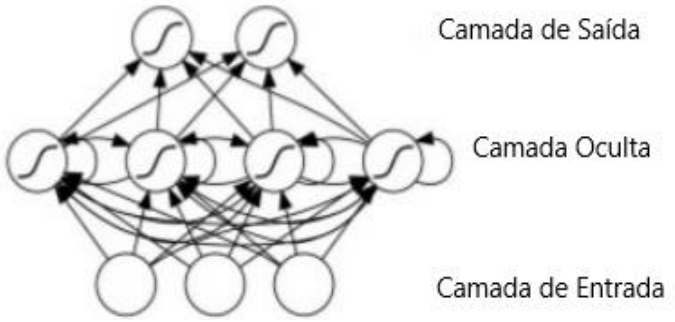
No modelo simplificado na Figura 1, as informações geradas pelos outros neurônios que constituem a rede entram em D entradas X_j no neurônio processador. O processamento constitui-se de uma combinação linear de entradas $net = W_1X_1 + W_2X_2 + \cdots + W_DX_D = \sum_{J=1}^{D} W_jX_J = \underline{W^TX}$, em que W_j representa pesos aplicados a cada valor de entrada que retratam a relevância da entrada X_j. Como resultado da combinação linear, obtém-se o valor de net, se tal valor exceder o limiar μ presente na função de ativação o neurônio resultará em 1 na saída binária y, caso não exceda o limiar a saída fica passiva em $y = 0$ (RAUBER, 2005).

Uma das características da RNA, assim como de outros algoritmos de aprendizagem de máquina, trata-se de que para o aprendizado seja efetivo, é importante que os dados de treino contenham instâncias distintas, para que a RNA possa "aprender" com acurácia os melhores pesos para produzir a saída desejada. Cada instância no conjunto de treino possui um rótulo, que denota a correta classificação da instância. Por exemplo, dado um problema para aprender a classificar um objeto como carro ou não-carro, precisa-se de carros para que possa ser extraída as suas características, tais como quantidade de rodas, tamanho, entre outros. Assim como instâncias de outros objetos que não sejam carros, para que a RNA possa diferenciá-las. Dessa forma, para que a aprendizagem seja efetiva, precisa-se primeiramente selecionar dados relacionados ao problema em questão. A partir desses dados, no processo de aprendizagem, a RNA analisa essas instâncias dos dados de treino para atualizar os seus parâmetros (pesos da RNA), de forma que o modelo seja efetivo para classificar as instâncias nos dados de treino, mas também instâncias que não estão presentes no conjunto de treinamento (NELSON, 2017).

No tocante à arquitetura, as redes neurais são definidas em camadas (Figura 2), que podem ser classificadas em três tipos, camada de entrada, camadas ocultas e camada de saída. Cada camada contém uma ou mais unidades, e as saídas das unidades são combinadas, fazendo

uso de funções de ativação (por exemplo, regressão logística), e usadas como entradas das unidades da camada posteriores (GANSSLE, 2018; NELSON, 2017).

Figura 2: Camadas de uma RNA



Fonte: NELSON, 2017.

As conexões entre as camadas da RNA podem se estabelecer entre unidades da mesma camada ou entre camadas posteriores, desta forma as informações fluem em diversos sentidos, e a saída da rede não possui dependência somente da entrada corrente, mas também de entradas anteriores (NELSON, 2017).

As RNAs apresentam diversas aplicações em áreas do conhecimento tais como, regressão, classificação e compactação de dados, em diversas aplicações tais como diagnósticos médico, exploração de dados de sobrevivência e em estudos epidemiológicos (SANTOS, 2005; DUH, 1998; LAPEER, 1995; RIPLEY, 1998; HAMMAD, 1996; EL-SOLH, 1999).

As redes neurais podem ser organizadas em diferentes arquiteturas, formando assim diversos tipos de RNAs. Uma dessas RNAs é a rede de aprendizagem profunda (do inglês deep learning), que está apresentando resultados muito positivos atualmente (LECUN et al., 2015). Ela utiliza o método de redes neurais artificiais de diversas camadas para aprender a representar e abstrair dados e até mesmo realizar atividades avançadas, que um especialista na área tenha dificuldade para desenvolver (LIN, 2018). Algumas redes neurais de aprendizado profundo são: Redes Neurais Recorrentes (RNN - ou *Recurrent Neural Network*) e as redes neurais Memória de Curto Prazo de Longa Duração (LSTM - ou *Long Short-Term Memory*). A rede LSTM proporciona uma preservação nas células de memória internas da rede RNN de serem atingidas pelo efeito de "aniquilamento" das funções de ativação logística em ligações recorrentes (SANTOS, 2018b).

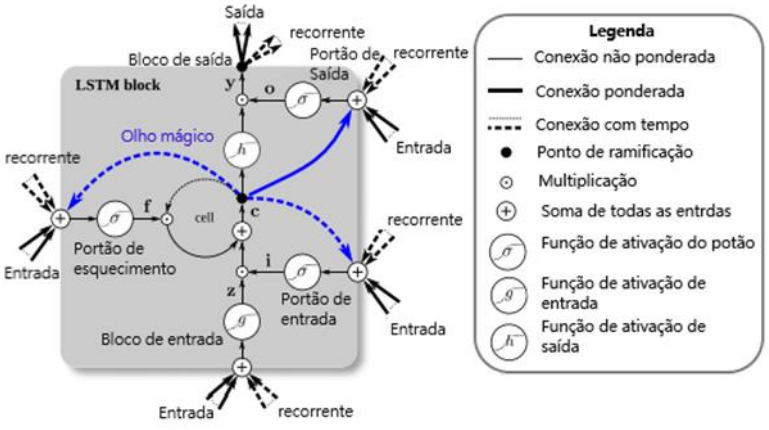
O modelo LSTM é um tipo de arquitetura de rede neural recorrente específica (RNN) que foi desenvolvido com o propósito de modelar sequências temporais e suas dependências de

longo alcance com mais precisão do que as RNN convencionais (SAK, 2014). Tal modelo apresentou uma execução melhor do que os RNNs na obtenção de conhecimentos de linguagens livres de contexto e sensíveis ao contexto (GERS, 2001).

A LSTM possui setores especiais denominados de blocos de memória presentes na camada oculta recorrente (Figura 3). Eles possuem células de memória com auto conexões guardando o estado temporal da rede, além de unidades multiplicativas próprias intituladas por portões de controle de fluxo de informações. Cada bloco de memória na arquitetura originária possui respectivamente um portão de entrada e um portão de saída. O portão de entrada tem como finalidade coordenar o fluxo de ativações de entrada na célula de memória, já o portão de saída comanda o fluxo de saída das ativações de células para o restante da rede (GERS, 1999).

Posteriormente, introduziu-se o portão de esquecimento nos blocos de memórias, tendo como propósito classificar o estado interno da célula antes de inseri-lo como entrada para célula por meio da conexão auto recorrente da célula, modificando adaptativamente a memória da célula. Além disto, a LSTM ainda conta com a *peephole connections* (conexões de olho mágico) de suas células internas que interliga os portões da mesma célula para que seja possível compreender o tempo específico das saídas geradas (GERS, 1999; GERS, 2002).

Figura 3: Bloco LSTM utilizado na camada oculta de uma rede neural recorrente



Fonte: GREFF, 2016.

A Figura 3 apresenta um esquema detalhado de um bloco LSTM, em que são apresentados três portões são eles: portão de entrada, portão de esquecimento e portão de saída; o bloco de entrada; três distintas funções denominadas de: função de ativação de saída, função de ativação de entrada e função de ativação do portão e por fim a conexão de olho mágico. A

conexão do bloco de saída é realizada de maneira recorrente retornando assim ao bloco de entrada e a todos os portões (GREFF, 2016).

2.2. Análise Léxica e Sintática

A análise léxica é a primeira etapa do processo de compilação. Possui a finalidade de realizar a seleção de itens léxicos ou lexemas aos quais denotam elementos da linguagem de programação, comumente denominados de *tokens*, que variam de acordo com sua estrutura. Alguns exemplos de lexemas ou *tokens* são os identificadores, palavras-chave, operadores, constantes, símbolos de pontuação, dentre outros elementos. Devido a questões de legibilidade dos analisadores léxicos e sintáticos, os *tokens* são referenciados por um código e a sua representação em forma de sequência de lexemas. (AHO, 2008; SEBESTA, 2018).

Atualmente, grande parte dos analisadores léxicos são constituídos de subprogramas que tem o propósito de localizar o próximo lexema na entrada, para determinar assim o seu código de *token*. Dentre as aplicações da análise léxica, inclui-se a detecção de comentários e espaços em branco excluindo-os da etapa de criação dos lexemas, encontrar erros sintáticos em *token*s informando-os aos usuários. O analisador léxico é ponto de partida para execução de um analisador sintático, ou seja, a análise léxica é uma entrada para a análise sintática (SEBESTA, 2018).

Por outro lado, o analisador sintático tem a finalidade de realizar uma verificação com intuito de definir se a cadeia de *tokens* é sintaticamente correta de acordo com a respectiva gramática. Algumas técnicas são utilizadas para criação do analisador sintático. Algumas delas se fundamentam na elaboração de uma árvore de derivações, em que as sequências de *tokens* podem ser elaboradas pela aplicação de alguma sequência de regras retiradas da gramática. Por outro lado, o método de análise sintático baseado em árvore sintática, podem utilizar exploração descendente, que edifica a árvore sintática das raízes até as folhas, ou exploração descendente que utiliza o procedimento oposto (SILVA, 2011).

2.3. Ferramenta ANTLR

O ANTLR em sua 4ª versão, é um eficiente criador de analisadores, tendo sua utilidade empregada na leitura, processamento, execução ou tradução de entradas na forma de texto ou arquivos binários. Possui aplicabilidade tanto na indústria como na academia, por exemplo na criação de diversos tipos de linguagens e ferramentas da engenharia de software. Na indústria foi empregada para o processamento de consultas do Twitter, no desenvolvimento da IDE SQL *Developer* e em ferramentas de análise de código como o C++ (PARR, 2013).

Seu funcionamento se dá a partir de uma gramática, ou seja, a partir de uma descrição formal da linguagem, o ANTLR cria um analisador para a linguagem que é capaz de produzir de maneira automática árvores sintáticas abstratas (do inglês *Abstract Syntactic Tree*). Árvores são estruturas de dados que retratam como a gramática corresponde aos valores de entrada. Além disso, o ANTLR cria automaticamente meios que possibilitam caminhar sobre as referidas árvores, possibilitando assim, a visita a cada nó e ante-nó dessas árvores com o propósito de executar o código correspondente do aplicativo (PARR, 2013).

A entrada do ANTLR é uma gramática livre de contexto, dentre outras informações contextuais. As gramáticas desenvolvidas em ANTLR fazem uso de uma sintaxe do tipo de *Yacc* integrada com operadores *extender* BNF (EBNF), com estrela Kleener (*) e por fim, literais de token dentro de aspas simples (PARR, 2011).

3. REVISÃO SISTEMÁTICA

Neste Capítulo apresenta-se uma revisão sistemática da literatura sobre o emprego de abordagens de aprendizado de máquina para ensino de programação. Ele está dividido em 8 seções, são elas: Problemas (Seção 3.1), Questões de Pesquisa (Seção 3.2), Estratégia de Pesquisa (Seção 3.3), Seleção de Estudos (Seção3.4), Extração das Informações (Seção 3.5), Resultados (Seção 3.6), Trabalhos selecionados (Seção 3.7) e Considerações Finais do Capítulo (Seção 3.8).

3.1. Problema

Diante da diversidade e evolução das linguagens de programação existentes atualmente, cada uma apresentando características e estruturas que buscam torná-las mais viáveis ao desenvolvimento de projetos e principalmente ao aprendizado, ainda há grandes dificuldades nas instituições de ensino superior, principalmente nas disciplinas introdutórias de programação, no tocante ao aprendizado dos alunos das linguagens de programação propostas por tais instituições.

Tal problema manifesta-se quando os alunos são submetidos a testes que requerem conhecimento básicos da linguagem, apresentando resultados insatisfatórios cometidos por faltas durante a escrita do código-fonte refletindo em erros durante a etapa de compilação e da deficiência de realizar a identificação e correção de tais erros que são notificados pelos compiladores. Tabanao et al. (2008) destaca que o texto simples pode torna-se um obstáculo para alunos que ingressam no estudo de programação. Santos et al. (2018) justifica que não há somente a necessidade de conhecer a semântica de uma linguagem de programação específica, mas é crucial aprender a inserir símbolos particulares de cada linguagem em ordem correta para que o computador tenha uma intenção coesa ao que lhe foi solicitado. Tabanao et al. (2008) categoriza o posicionamento correto dos símbolos – característicos de cada linguagem de programação – de sintaxe, ao qual é o erro mais cometido pelos iniciantes de programação e por programadores experientes.

Para fins de ajudar alunos ou até mesmo programadores experientes durante a escrita de código-fonte de seus trabalhos ou projetos, algumas técnicas foram propostas na literatura.

Por exemplo, Becker (2015) apresenta um estudo de caso com intuito de investigar os resultados obtidos por alunos ao utilizarem um software de sua autoria, que auxilia os alunos a vencerem suas dificuldades em encontrar e corrigir erros elencados pelo compilador. O software possui como característica principal o aprimoramento nas descrições das mensagens de erros, em que são apresentadas de maneira direta e informativa.

Para entender melhor o uso de ferramentas computacionais para o ensino de programação, neste trabalho, foi realizada uma revisão sistemática das técnicas que utilizam processamento de linguagem natural para auxiliar estudantes/programadores na resolução de tarefas de programação. O objetivo deste trabalho é investigar as abordagens presentes na literatura que englobam as ferramentas de detecção e correção de erros de sintaxe por meio de processamento de linguagem natural. O resultado almejado é uma descrição do maior número de técnicas presentes atualmente que utilizam aprendizado de máquina para ajudar estudantes/programadores nas tarefas de programação. Deseja-se entender por exemplo:

- Abordagem de aprendizagem de máquina empregada;
- Tipo de estudantes (novatos, experientes, online);
- Tipos de erros mais comuns que são alvos da abordagem;
- Linguagem de programação utilizada;
- Paradigma de programação (funcional, orientado a objeto (OO), imperativo);
- Tipo de saída (fornecer dicas, corrigir automaticamente);
- Plataforma empregada (Web, Mobile, Desktop).

3.2.Questão de Pesquisa

Diante da problemática apresentada, emergiram duas questões de pesquisa, são elas:

RQ1 – Quais as técnicas empregadas para auxiliar estudantes na resolução de tarefas de programação, que utilizam aprendizado de máquina (Processamento de Linguagem Natural)?

RQ2 - Quais os métodos que possibilitam a identificação e correção de erros sintáticos?

3.3.Estratégia de Pesquisa

Como estratégia de pesquisa, tem-se a seguinte:

a) Derivar a maior parte dos termos usando as características: população, intervenção e saída das questões de pesquisa;
b) Identificar sinônimos para os termos encontrados;
c) Checar palavras-chaves de artigos disponíveis;
d) Quando a base de dados permitir usar conectores booleanos (e.g., and e or);
e) Usar a pesquisa padrão das ferramentas de busca.

Os resultados para os itens a) e b) da estratégia de pesquisa foram sumarizados na Tabela 1.

Tabela 1: Separação da questão de pesquisa em intervenção, população e saída, junto com conjunto de sinônimos

Sumário da Decomposição da Questão de Pesquisa e Sinônimos		
Característica	Valor (inglês)	Lista de sinônimos (em inglês)
Intervenção	Processamento de Linguagem Natural (*Natural language processing*)	• *Machine Learning*
População	Desenvolvedor de Software (*developer*)	• *Programmer* • *Student*
Saída	Código-Fonte Correto (*Correct Source Code*)	• *Implementation* • *Syntactic Errors*

Fonte: Próprio Autor (2020).

Na Tabela 1 os termos elencados nas características intervenção, população e saída foram extraídos das questões de pesquisa RQ1 e RQ2. A categoria intervenção define qual intervenção será investigada na revisão sistemática, a característica população apresenta os termos referentes a população incluída na pesquisa e a característica saída refere-se aos desfechos investigados.

Com a finalidade de encontrar palavras-chaves, a seguir, é apresentado alguns estudos que serviram como base para encontrar termos focados no campo apresentado na problemática.

- *Automated Correction for Syntax Errors in Programming Assignments using Recurrent Neural Networks;*
- *DeepFix: Fixing Common C Language Errors by Deep Learning;*
- *LL(*): The Foundation of the ANTLR Parser Generator;*
- *Syntax and Sensibility: Using language models to detect and correct syntax erros.*

Os termos novos encontrados foram os seguintes:

- *Artificial Intelligence*
- *Deep learning*
- *Programming education*
- *Program repair*
- *Recurrent neural networks*

A seguir pode ser visto a *string* de pesquisa utilizada para realizar a busca por estudos na ACM. As strings de busca para as outras fontes podem ser vistas no Apêndice A.

content.ftsec:(+("Recurrent neural networks" "Artificial Intelligence" "Machine Learning" "Deep learning" "Natural language processing") +(Programmer student developer) +("Program repair" "Programming education" "Syntactic Errors"))

Os critérios para seleção nas bases de dados encontram-se sumarizados na Tabela 2.

Tabela 2: Critério de Seleção de Fontes

Seleção de estudos	
Linguagem	Inglês
Método	Usando engenho de pesquisa da lista de base de dados
Base de dados	IEEE *Explore* ACM *Digital Library* Scopus
Revisão	Autores

Fonte: Próprio Autor (2020).

A *string* de pesquisa básica é descrita abaixo. A lista customizada da string para cada base de dados pode ser vista no Apêndice A.

("Recurrent neural networks" OR "Artificial Intelligence" OR "Machine Learning" OR "Deep learning" OR "Natural language processing") AND (Programmer OR student OR developer) AND ("Program repair" OR "Programming education" OR "Syntactic Errors")

3.4.Seleção de Estudos

Nesta seção apresenta-se os critérios e os procedimentos adotados para a seleção dos estudos almejados desta revisão sistemática.

Os critérios para seleção de estudos estão sumarizados na Tabela 3. A elaboração de tais critérios se deu mediante à análise das questões de pesquisa, bem como a diversidade de trabalhos presentes na literatura, onde buscou-se sintetizar os principais artigos voltados ao tema e mais recentes trabalhos.

Tabela 3: Critérios Para Seleção de Estudos

Seleção de estudos	
Critérios de exclusão	O artigo é relacionado a técnicas de processamento de linguagem natural. O artigo é externo ao tema de pesquisa. Artigos ou periódicos. Artigo publicado entre 2015 a 2019.
Critérios de inclusão	O artigo está relacionado a auxiliar estudantes em tarefas de programação. Responde à questão de pesquisa. O artigo é relacionado a técnicas de correção e localização de erros de sintaxe.
Tipos de estudos	Estudos empíricos, teóricos e secundários
Múltiplos estudos	Selecionar a versão mais completa

Fonte: Próprio Autor (2020).

A etapa de seleção de estudos realiza-se por meio da utilização do software StArt, que dispõem funcionalidades que auxiliam o pesquisador durante o processo de elaboração da revisão sistemática da literatura. Inicialmente criou-se um projeto ".start" no referido software, onde foram preenchidos todos os campos referentes ao protocolo da revisão sistemática. Em seguida, realizou-se uma busca avançada nas bases de dados da IEEE, ACM e Scopus em que as strings de buscas foram formatadas de acordo com os padrões exigidos por cada base de dados.

Após a busca, realizou-se em cada base de dados o *download* de um arquivo ". bibtex" que contém informações dos artigos retornados pela busca. Os referidos arquivos foram exportados para o software StArt[1], onde apresentou em forma de lista, todos os artigos e suas principais informações. O software disponibiliza quatro categorias na etapa de seleção, aos quais os artigos podem se elencar, são eles: "*Unclassified Papers*", "*Duplicated Papers*", *"Rejected Papers*" e "*Accepted Papers*".

Por padrão todos os artigos exportados são classificados como *Unclassified Papers*, em seguida, realizou-se a leitura do título de cada artigo. Se o artigo se enquadrou dentro de um critério de exclusão, selecionou-se e moveu artigo para a categoria *Rejected Papers*. Caso

[1] http://www.dc.ufscar.br/~lapes/start/InstallStArt2.3.4.2.exe.

contrário, selecionou-se e moveu artigo para a categoria *Accepted Papers*. Os artigos duplicados foram movidos para categoria *Duplicated Papers*.

Por fim, realizou-se a etapa de extração dos dados onde é feita a leitura completa de todos os artigos elencados como *Accepted Papers* na etapa de seleção. Se o artigo se enquadra dentro de um critério de inclusão colocou-se o artigo na seção *Accepted Papers,* caso contrário o artigo e movido para categoria *Rejected Papers*.

3.5. Extração de Informações

Para cada artigo elencado como *Accepted Papers* na etapa de seleção, realizou-se a extração das informações por meio da leitura completa dos artigos elencados. Os mesmos foram organizados em uma tabela desenvolvida no programa Excel, constando em cada coluna informações referentes a ID do *paper*, título do artigo, autores, breve resumo do trabalho, detalhes da publicação, abordagem de aprendizagem de máquina, tipos de estudantes, tipos de erros mais comuns que são alvos da abordagem, linguagem de programação utilizada, paradigma de programação, tipo de saída e plataforma empregada. Tendo, o ambiente para extração de informações definido, os autores iniciaram a extração das informações dos trabalhos selecionados.

3.6. Resultados

Esta seção apresenta os resultados alcançados pela pesquisa. O processo de pesquisa resultou num total de 34 trabalhos pré-selecionados. Com base nos critérios de inclusão e exclusão, foram excluídos 13 trabalhos, totalizando assim 21 trabalhos que apresentam resultados relevantes para processo de extração de dados. A Tabela 4 expõem os resultados do processo de pré-seleção e inclusão, juntamente com as bases de dados aos quais os trabalhos pertencem.

Tabela 4: Resultado do Processo de Pesquisa

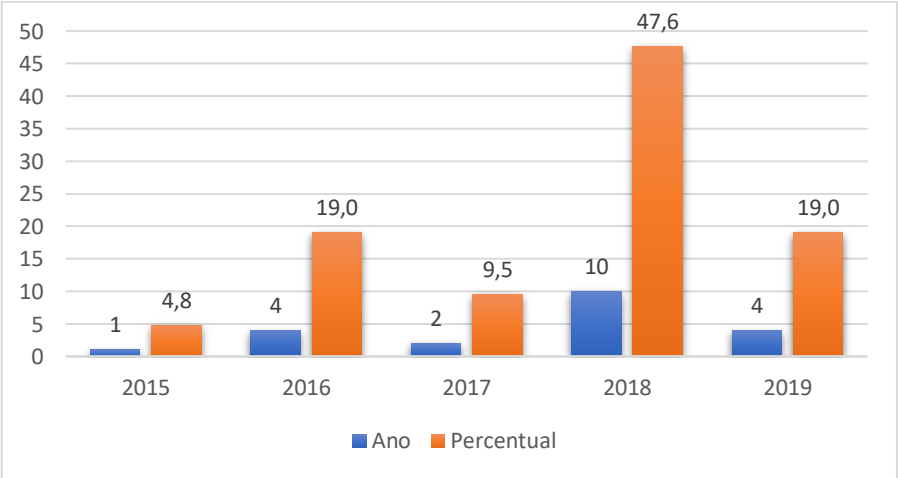
Base de Dados	Artigos Pré-Selecionados	Artigos Inclusos
ACM	10	9
IEEE	4	3
SCOPUS	20	9
Total	**34**	**21**

Fonte: Próprio Autor (2020).

Durante a etapa de extração de informações e de seleção dos trabalhos pré-selecionados, o critério de seleção adotado foi que os trabalhos estarem relacionados a auxiliar estudantes em tarefas de programação. Os demais artigos não incluídos voltavam-se a temas opostos aos critérios apresentados por esta revisão, por exemplo: uma aplicação que fornece um conjunto de problemas para auxiliar estudantes a aprender determinado conceito relacionado a programação; abordagem de atribuição de nota em tarefas de programação; abordagens de ensino-aprendizagem para ensinar programação para cursos que não são voltados a computação.

O Gráfico 1 apresenta a disposição anual dos trabalhos selecionados. Os dados apontam um crescimento considerável nos últimos dois anos (2018 e 2019) a respeito do desenvolvimento de pesquisas no tocante ao tema estudado, referente a um percentual de 66,6%, tendo o ano de 2018 o maior percentual de trabalhos entre os demais, com 47,6% de trabalhos publicados. Por outro lado, o ano de 2015 apresentou dentro do período pesquisado o mais baixo índice de publicações com 4,8%.

Gráfico 1: Disposição Anual dos Trabalhos

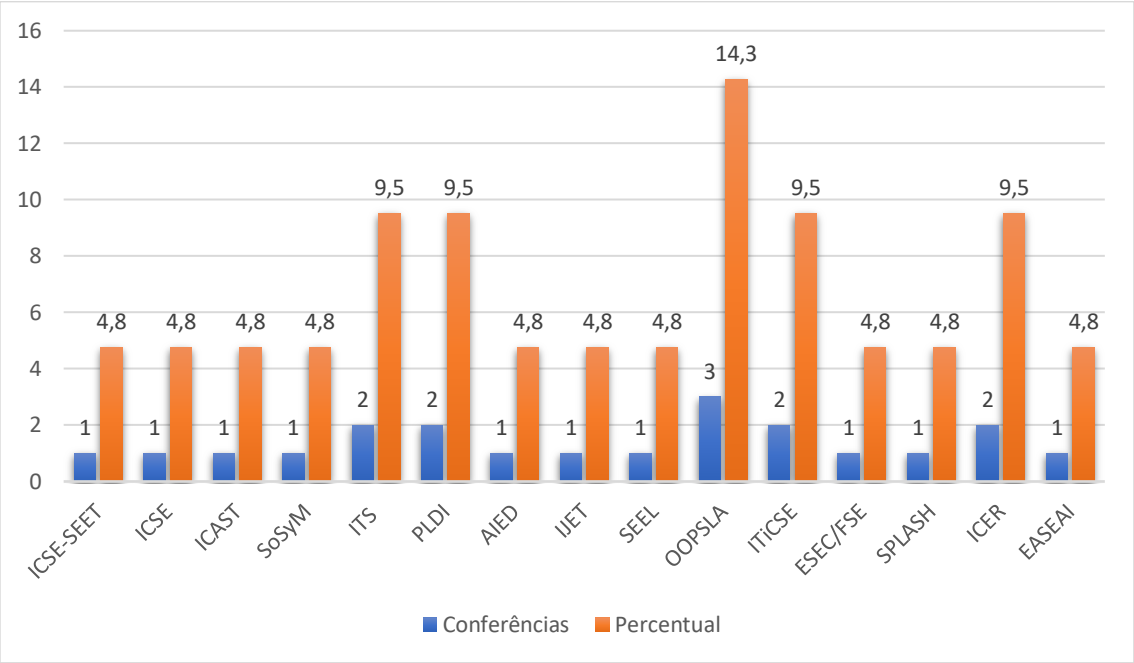


Fonte: Próprio Autor.

O Gráfico 2 apresenta a distribuição detalhada das conferências, jornais e eventos aos quais os trabalhos selecionados foram publicados. A partir dos dados coletados é possível observar o quão numeroso a quantidade de eventos, jornais e conferências aos quais torna-se viável a realização de publicações que envolva o tema da pesquisa em questão, totalizando assim, a quantidade de 15 eventos distintos, são eles:

- *International Conference on Software Engineering: Software Engineering Education and Training* (ICSE-SEET);
- *International Conference on Software Engineerin*g (ICSE);
- *International Conference on Advancements of Science and Technology* (iCAST);
- *International Journal on Software and Systems Modeling* (SoSyM);
- *Intelligent Tutoring Systems* (ITS);
- *Programming Language Design and Implementatio*n (PLDI);
- *International Conference on Artificial Intelligence in Education* (AIED);
- *International Journal of Engineering and Technology* (IJET);
- *Smart Education and E-Learning* (SEEL);
- *Object-Oriented Programming, Systems, Languages, and Applications* (OOPSLA);
- *Innovation and Technology in Computer Science Education* (ITiCSE);
- *European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (ESEC/FSE);
- *Systems, Programming, Languages, and Applications: Software for Humanit*y (SPLASH);
- *International Computing Education Research* (ICER);
- *Education through Advanced Software Engineering and Artificial Intelligence* (EASEAI).

Gráfico 2: Distribuição das Conferências, Jornais e Eventos dos Trabalhos Publicados

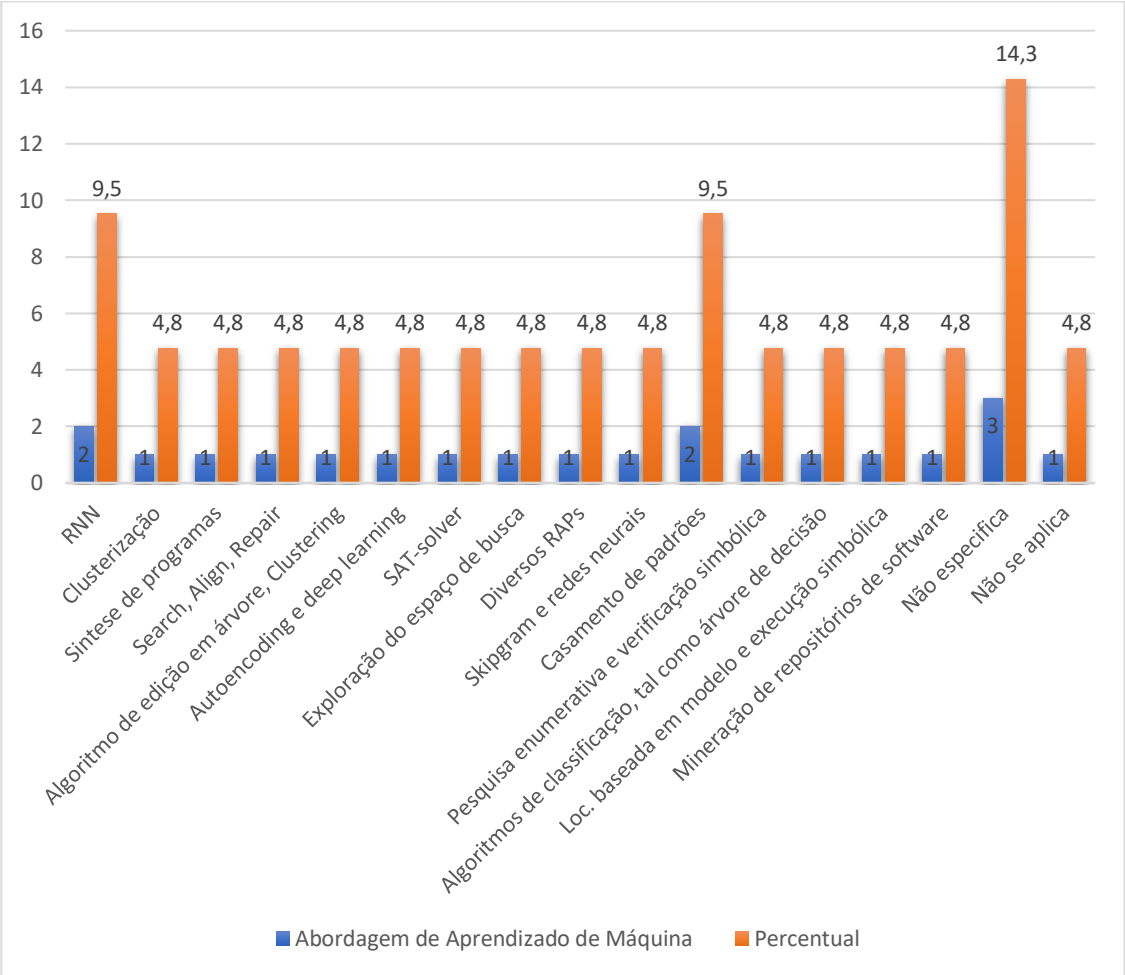


Fonte: Próprio Autor.

Diante dos eventos, jornais e conferências apresentados, o que apresentou maior percentual de publicações foi OOPSLA com 14,3%, juntamente com ITS, PLDI, ITiCSE e ICER ambos com o percentual de 9,5%, os demais obtiveram um percentual de 4,8%, a partir dos presentes valores é possível constatar uma boa distribuição entre os trabalhos publicados nos distintos meios apresentados.

As abordagens de aprendizado de máquina adotadas e descritas nos trabalhos aqui pesquisados estão sintetizadas no Gráfico 3.

Gráfico 3: Abordagens de Aprendizado de Máquina



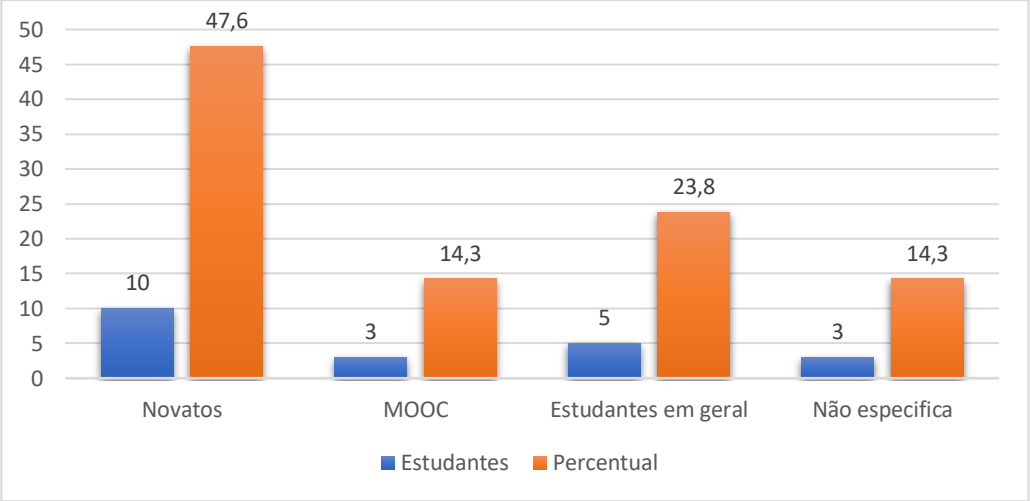
Fonte: Próprio Autor.

Foram utilizadas 15 técnicas de aprendizado de máquina distintas, dentre elas as mais empregadas são as RNNs e Casamento de padrões, ambas apresentando um percentual de 9,5%, as demais abordagens apresentam mesmo percentual de utilização de 4,8%. Em alguns trabalhos não realizou-se a especificação do tipo de abordagem de aprendizado de máquina utilizadas, apresentado um percentual de 14,3% e a não adoção da referida abordagem constou-se um percentual de 4,8%, o mesmo inclui-se como trabalho válido para este fim de pesquisa, devido apresentar uma abordagem que contribuía para ajudar a alunos na resolução de tarefas de programação.

A partir das informações apresentadas é perceptível a diversidade de técnicas e métodos de aprendizado de máquina presente na literatura, que possuem a finalidade de auxiliar os alunos de programação na resolução de tarefas, bem como, desenvolver suas competências na área de programação, contribuindo assim, para crescimento intelectual dos mesmos.

Os tipos de estudantes ao qual as abordagens de aprendizado de máquina foram destinadas, encontram-se sumarizados no Gráfico 4.

Gráfico 4: Tipos de Estudantes



Fonte: Próprio Autor.

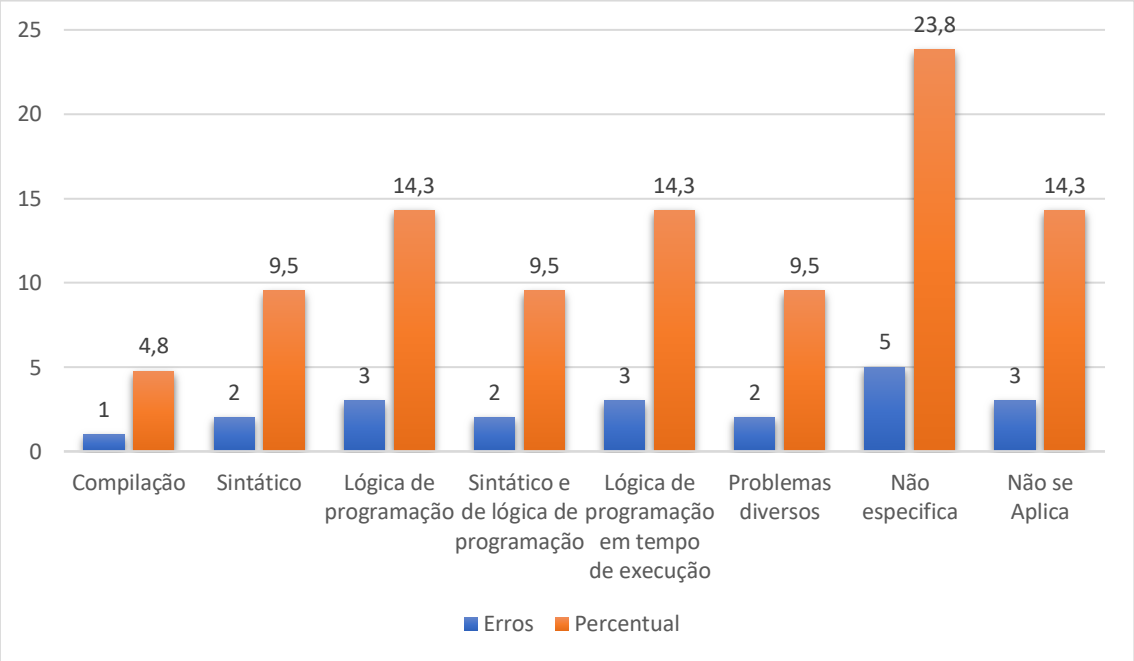
A abordagem predominante atribui-se a estudantes novatos de programação, que apresenta percentual de 47,6%, já as abordagens destinadas para estudantes em geral, independentemente de seu nível de familiaridade e agilidade com as diferentes linguagens de programação, consta um percentual de 23,8%. Os estudantes que utilizam os *Massive Open Online Course* (MOOC) – ambiente virtual de aprendizagem de cursos abertos – evidência um percentual de 14,2%. Dentre os trabalhos, 14,2% não especifica para quais tipos estudantes é destinado a abordagem desenvolvida.

A partir destes dados, é possível observar a existência da preocupação e esforços dos pesquisadores e desenvolvedores no processo de aprendizagem dos estudantes de programação sejam eles novatos ou não, para que os mesmos possam evoluir no processo de aprendizagem e reduzir o número de desistências do estudo de programação. Ainda é possível observar a expansão da aplicabilidade das abordagens de aprendizado de máquina, que se fazem presente em meios virtuais de ensino como os MOOCs.

Os tipos de erros mais comuns que são alvos das abordagens dos trabalhos resultados do processo de pesquisa, encontram-se expostos no Gráfico 5. Os trabalhos resultantes da revisão sistemática em sua maioria não especificam para quais tipos de erros as ferramentas desenvolvidas são destinadas a solucionar, atingindo um percentual de 23,8%, entretanto 14,3% dos trabalhos apresentam correção erros voltados a lógica de programação, bem como, a

correção dos mencionados erros durante o processo de compilação constando também um percentual de 14,3%. Por outro lado, os erros sintáticos e de lógica de programação, juntamente com erros diversos, ambos obtiveram o percentual de 10%. Os erros de compilação constaram um percentual de 5% e os sintáticos de 9%. A não aplicabilidade de ferramentas destinadas correção de erros atingiu um percentual de 14,3% tais ferramentas tornam-se válidas devido prestar auxílio aos estudantes durante o processo de solução de problema, como por exemplo, a geração de *feedbacks* explicativos que ajudam os alunos a solucionarem os problemas.

Gráfico 5: Erros que são alvos das abordagens



Fonte: Próprio Autor.

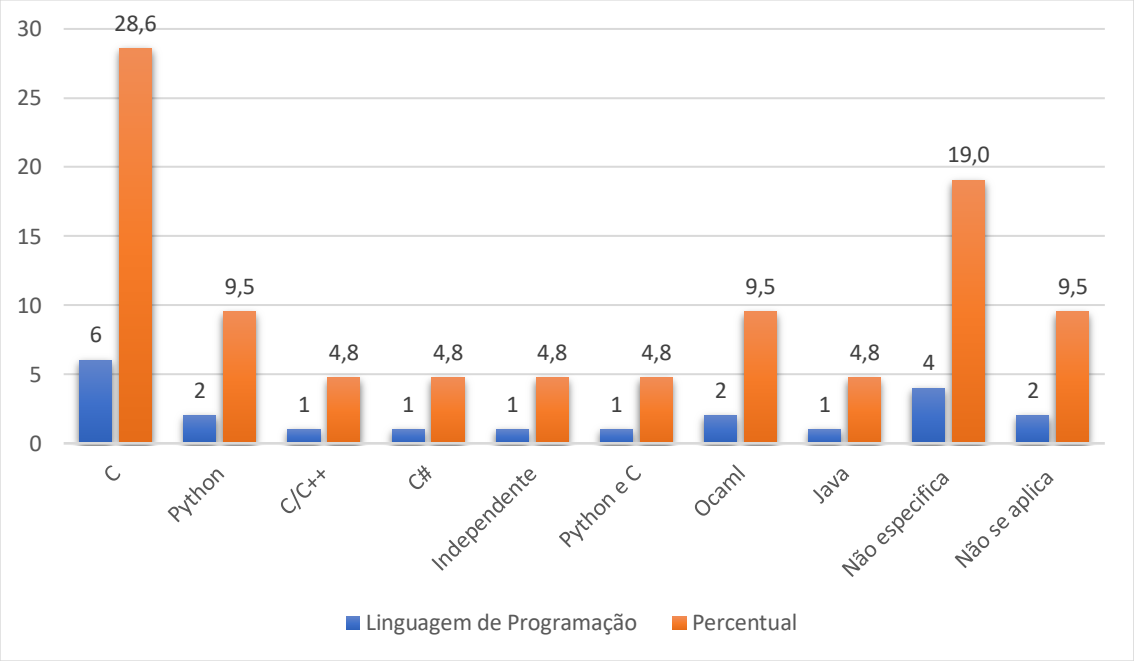
A partir dos dados é possível observar a variedade de ferramentas destinadas a solucionar e a prestar auxílio a diferentes problemas relacionados a erros durante o desenvolvimento de programas. Dispondo aos estudantes diferentes tipos de suporte para quando não encontrar uma solução viável para seus erros.

O Gráfico 6 apresenta os tipos de linguagens de programação utilizadas para o desenvolvimento das ferramentas expostas pelos trabalhos resultantes desta revisão sistemática. A linguagem de programação C, apresenta-se com maior percentual de utilização com 28,6%, sua elevada adoção é consequente de sua capacidade de ser uma linguagem leve para o ser processada. As linguagens Python e Ocmal ambas apresentaram um percentual de 9,5% de utilização. Os trabalhos que não especificam a linguagem de programação adotada, atingindo

um percentual de 19%. Por outro lado, a não aplicação de linguagens de programação apresentou um resultado de 9,5%, tais abordagens são destinadas a identificação de técnicas que influenciam no aprendizado dos estudantes e um *survey* relacionados a sistemas de tutoria inteligente.

Os dados refletem uma elevada adoção de diferentes tipos de linguagens de programação para a construção de ferramentas que possibilitam ajudar os alunos a progredirem no aprendizado de programação.

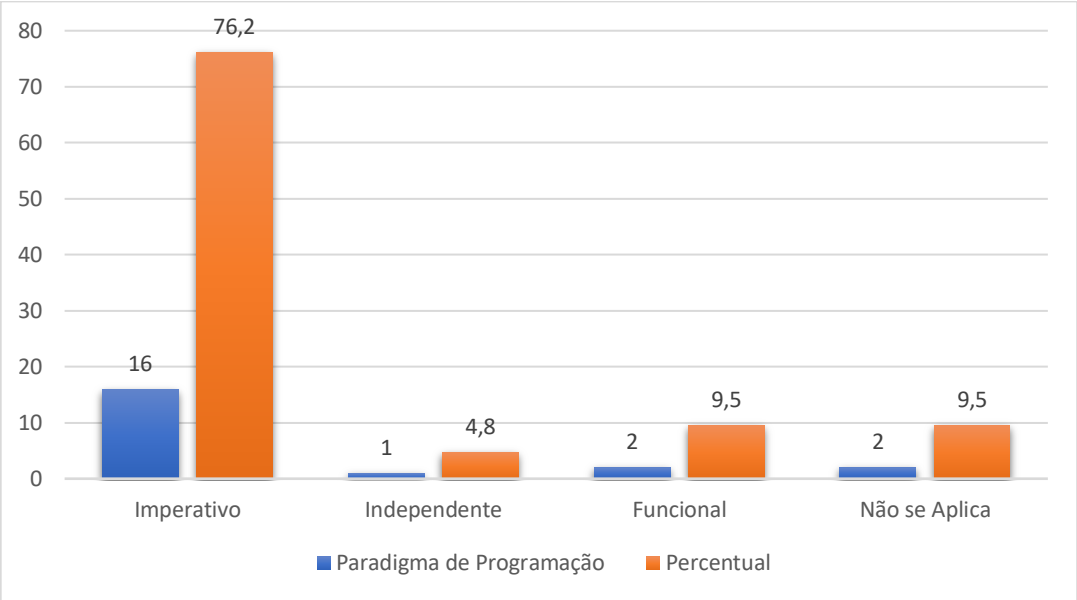
Gráfico 6: Linguagens de Programação Utilizadas



Fonte: Próprio Autor.

Os diferentes tipos de paradigmas de programação utilizados pelos trabalhos, encontram-se sintetizados no Gráfico 7. A adoção do paradigma imperativo sobressaiu-se com relação aos demais, atingindo um percentual de 76,2%. Por outro lado, o paradigma funcional apresentou a segunda maior utilização com percentual de 9.5%. A não aplicação do paradigma obteve o percentual de 9,5% tal valor é decorrente da não utilização de linguagens de programação como apresentado no gráfico anterior (Gráfico 6).

Gráfico 7: Paradigmas de Programação



Fonte: Próprio Autor (2020).

Os tipos de saídas resultantes da pesquisa que as ferramentas disponibilizam para os alunos para fins de auxiliá-los durante a resolução de tarefas estão sintetizados na Tabela 5.

Tabela 5: Tipos de saídas apresentadas pelas ferramentas

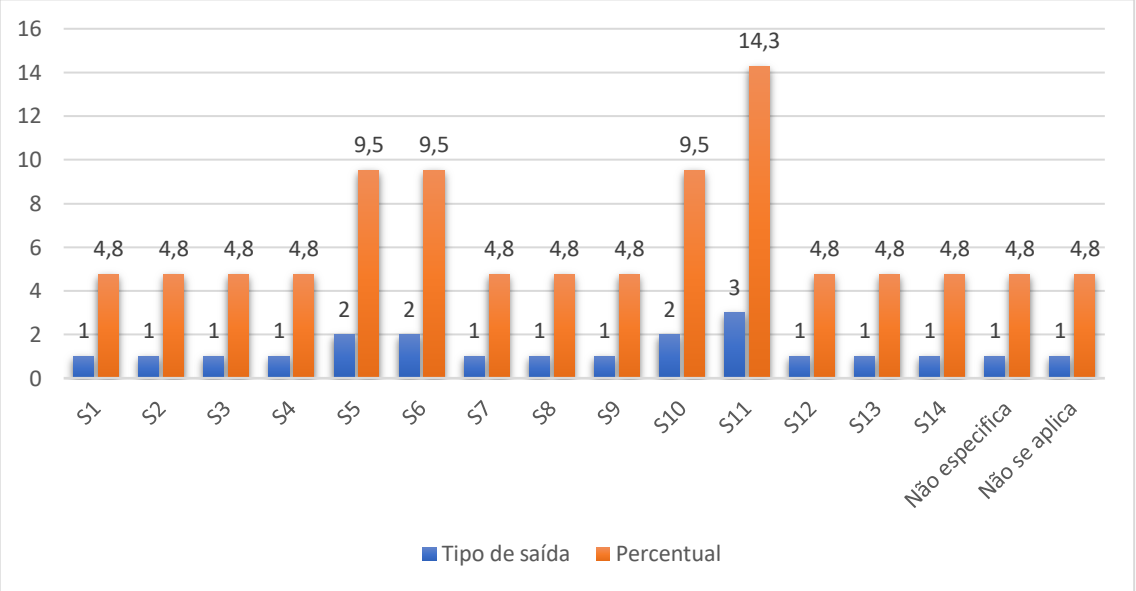
Identificador	Tipos de saídas
S1	Recomendações concretas e abstratas de correção de código
S2	Correção Automática de programas
S3	Feedback sobre erros de lógica de programação
S4	localizações que podem conter faltas
S5	Dicas
S6	Sugestão de correções
S7	*Feedback*
S8	Explanação de *bugs*
S9	Lista de localizações que podem conter faltas
S10	Correção de programas
S11	Reparo de programas
S12	Identificação de estudantes propensos a resultados ruins/bons
S13	Proposição de métricas para identificar o perfil dos estudantes
S14	Identificar quando o desenvolvedor está com problema de aprendizado

Fonte: Próprio Autor.

Na Tabela 5 são apresentados 14 tipos distinto de saída apresentados pelas ferramentas que encontram-se vinculadas a um identificar com o objetivo de facilitar o processo de interpretação das respectivas saídas na plotagem do Gráfico 8.

O Gráfico 8 expõem os diferentes tipos de saídas (Tabela 5) que as ferramentas disponibilizam para os alunos para fins de auxiliá-los durante a resolução de tarefas. O reparo de programas (S11) foi o tipo de saída mais comum entre as abordagens apresentando um percentual de 14,3%, já os tipos de saídas relacionadas a dicas (S5), sugestão de correção (S6) e correção de programas (S10), ambos apresentaram um percentual de 9,5%. Os demais tipos de saídas obtiveram o mesmo percentual, com o valor de 4,8%. Diante dos dados, é possível constar as diferentes abordagens que possibilitam os estudantes de programação a terem um maior suporte para sanar seus erros e dúvidas.

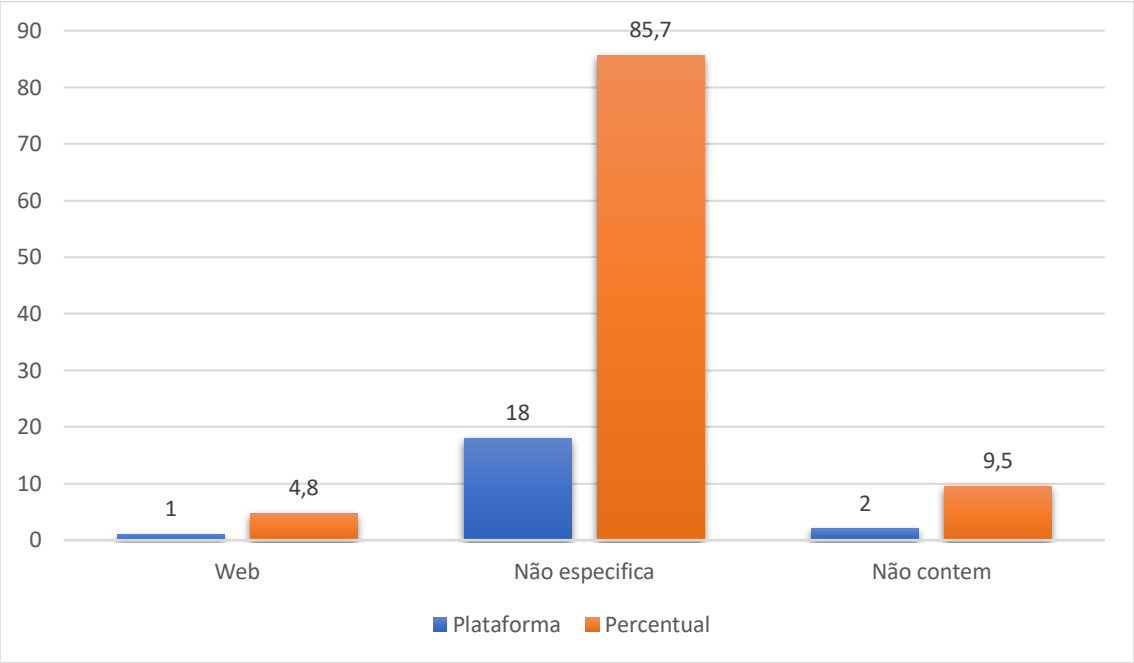
Gráfico 8: Tipos de Saída



Fonte: Próprio Autor.

As plataformas aos quais as abordagens desenvolvidas são destinadas estão sintetizadas no Gráfico 9. Na maior parte dos trabalhos, não realizou-se a especificação de qual plataforma a ferramenta proposta iria atuar, obtendo um percentual de 85,7%. Apenas 4,8% destina-se a aplicação de uma plataforma Web, enquanto 9,5% não dispõem de uma plataforma.

Gráfico 9: Plataforma Empregada



Fonte: Próprio Autor.

3.7.Trabalhos Selecionados

Neste capítulo apresenta-se uma breve discussão a respeito dos trabalhos resultantes da revisão sistemática. Os trabalhos aqui apresentados são referenciados pelos seus devidos identificadores, como por exemplo: P01, P02 e P03. A lista completa com todas as informações de cada trabalho pode ser encontrada no Apêndice B.

O trabalho P01 apresenta um sistema, TRACER, que é capaz de corrigir erros de compilação em programas redigidos por estudantes. Um dos motivos para o desenvolvimento deste trabalho é que erros de compilação gerados e apresentados para os estudantes não são simples de serem entendidos. A abordagem localiza o erro de compilação no código e fornece recomendações de como corrigir o código.

Diferentemente de P01, o trabalho P02 apresenta uma abordagem para realizar reparos sintático de programas que utiliza uma abordagem neuro-simbólica, que combina Redes Neurais Recorrrentes (RNN) e programação baseada em restrições. Os autores apontam que muitas abordagens são baseadas em Árvore Sintática Abstrata (AST) do Programa, que não pode ser construída se o programa contém erros sintáticos. Nessa abordagem, os programas são

corrigidos (reparados) usando RNN e posteriormente o programa é reparado novamente usando uma abordagem baseada em restrições formais a garantir a corretude do programa.

A abordagem descrita pelo trabalho P03 é relacionada a uma ferramenta para detectar erros de lógica de programação em submissões para juízes *online*. Os juízes *online*, usualmente, oferecem *feedback* mínimo para as submissões informando se o programa foi aceito ou não. Dessa forma, o usuário ainda precisa de uma explicação de porque o seu programa foi rejeitado pelo juiz *online*. No trabalho, uma técnica é proposta para detecção de erros de lógica de programação baseado na estrutura de um programa incorreto e o programa correto, assim como a similaridade entre o código correto e o código incorreto.

O trabalho P04 apresenta uma abordagem para identificação de localizações em tarefas de programação dos estudantes que podem conter faltas. A abordagem recebe como entrada uma *suite* de teste, e usa execução simbólica com o objetivo de identificar localizações no programa que contém faltas e que uma possível solução para o problema existe.

Uma abordagem de sistema de tutoria inteligente é descrita pelo trabalho P5, tal abordagem possui o objetivo de prover auxílio aos estudantes durante a resolução de tarefas de programação. A abordagem é capaz de promover *feedback* para os estudantes enquanto estão realizando a codificação, sem que o código esteja necessariamente completo, de forma a auxiliar os estudantes a finalizar as suas soluções para as tarefas de programação. A abordagem utiliza três tipos de recursos para prover auxílio: vídeo, dicas de programação e fluxograma.

Apresentando algumas semelhanças com a abordagem descrita no trabalho anterior, o trabalho P6 apresenta uma técnica para fornecer *feedback* para estudantes que estão resolvendo exercícios de programação. Para isso, o trabalho utiliza uma abordagem denomina "*Search, Align e Repair*", descrito como um *framework* baseado em dados para realizar reparos de programas. A abordagem primeiramente recebe um programa incorreto e uma solução para o programa. Em seguida, ela utiliza a Árvore Sintática Abstrata do programa (AST) para alinhar as declarações do programa incorreto com o programa correto. Finalmente, a técnica sugere resoluções para o programa incorreto.

O trabalho P7 também apresenta uma técnica para promoção de *feedback* denominada de TipC. A técnica utiliza método de edição de árvore sintática para identificar possíveis soluções corretas para um problema incorreto fornecido. Baseado nessa análise, a técnica sugere correções no código incorreto para resolver problemas de lógica no programa do estudante. A

técnica também ajuda os professores fornecendo uma ferramenta de visualização, em que o instrutor pode observar os padrões de erros nos programas dos estudantes.

Uma abordagem em desenvolvimento, do uso de sistema de tutoria inteligente para prover *feedback* a estudantes similares aos produzidos por humanos é tratado no trabalho P8. A abordagem proposta almeja criar modelos para representar a resposta incorreta assim como a resposta correta para produzir *feedback* para o estudante da forma mais apropriada. Por outro lado, o trabalho P9 expõe um *survey* sobre os sistemas de tutoria inteligente baseados em geração de *feedback* produzidos por dicas.

A respeito dos aspectos que influenciam no aprendizado dos estudantes, o trabalho P10 apresenta uma abordagem que realiza uma identificação dos referidos aspectos. Para isso, o artigo utiliza mineração de dados para determinar as propriedades que possuem impacto no aprendizado dos estudantes para determinar o padrão de comportamento singular dos estudantes.

O trabalho P11 demonstra uma abordagem que provê explanação automática de problemas encontrados em submissões de estudantes. Para fornecer a explanação, a abordagem recebe como entrada, uma submissão incorreta fornecida pelo estudante e uma versão correta do programa, fornecida pelo instrutor e um caso de teste em que o programa submetido pelo estudante falha. A partir desses dados, a abordagem produz um relatório contendo problemas que se apresentam na submissão incorreta. Para fornecer esse *feedback*, a abordagem utiliza diferentes abordagens, dentre elas a modelagem do problema para ser resolvido com um SAT-*solver*.

Uma abordagem rápida e acurada para realizar a detecção de faltas em programas escritos em C, é a abordagem proposta pelo trabalho P12. A técnica realiza sua tarefa automaticamente e usa a *suite* de teste associada ao programa com faltas para elencar as localizações no código em que uma falta no programa pode ser reparado, de forma a ser aceito em todos os casos de teste da *suite* de testes fornecida. Para isso, a abordagem faz uso de uma abordagem para localização de faltas baseada em modelos.

No trabalho P13 é realizado uma investigação a respeito do uso de Rapadores Automáticos de Programas (RAP) para auxiliar a resolver problemas de programação. Um RAP é uma abordagem que resolver faltas em programas que vem sendo largamente aplicados. Quatro RAPs (HenProg, AE, Angelix e Prophet) foram aplicados em 661 programas escritos por estudante em cursos de programação introdutória.

Uma abordagem denominada sk_p, é proposta pelo trabalho P14 para corrigir programas no contexto de Cursos Livres *Online* Massivos (CLOM). A abordagem é capaz de corrigir, tanto problemas relacionados a sintaxe e a lógica de programação do problema. Para isso, a abordagem usa um método de síntese de programas (i.e., uma abordagem para geração automática de programas). A abordagem usa diferentes abordagem de aprendizagem de máquina, dentre eles *skiptogram* e modelos de rede neural seq2seq.

O trabalho P15 que, encontra-se ainda em fases de andamento a respeito da aplicação de aprendizagem de máquina para auxiliar na aprendizagem de programação por novatos. A proposta do artigo é a identificação de estudante que estão sobre risco, no intuito de fornecer sugestões de como eles podem melhorar a sua forma de programar.

Uma abordagem, denominada de CLARA é apresentada no trabalho P16. Ela é utilizada para o reparo automático de programas para tarefas de programação introdutória. Para reparar os programas autenticamente, a abordagem usa as soluções existentes para o problema de forma a corrigir as submissões incorretas. Para isso, propõe-se um método de clusterização para encontrar casamentos entre as soluções corretas e incorretas.

Dispondo a mesma finalidade do trabalho P16, o trabalho P17 exibe uma técnica para reparo automático de programas para linguagens funcionais. Especificamente, a técnica foca-se em gerar um conjunto de testes que seja capaz de diferenciar um programa referência (solução correta) de um programa incorreto. Para realizar essa atividade, a técnica recebe como entrada o programa referência e a submissão incorreta do estudante. Baseada nesses dados, a técnica automaticamente gera casos de testes que capturam a diferença semântica entre os dois programas, usando uma técnica que combina pesquisa enumerativa e verificação simbólica.

O trabalho P18 apresenta uma abordagem para identificação de estudantes que precisam de assistência. A técnica utiliza diferentes abordagens para classificação de estudantes, dentre eles a combinação de pedaço de código, criados por estudantes e abordagens de aprendizagem de máquina.

Um conjunto de aspectos que podem ser utilizados para criar o perfil de novatos na resolução de problemas de programação, é o tema que retrata-se no trabalho P19. O artigo nota que várias métricas para caracterização do perfil dos estudantes são dependentes do contexto. Adicionalmente, o trabalho revisa as métricas que são utilizadas para criar o perfil dos estudantes e propõe uma abordagem que pode ser utilizada para o desenvolvimento de uma nova métrica.

O foco do trabalho P20 é apresentar um sistema, denominado de FixML, para reparo de programas para linguagens funcionais. Para realizar o reparo de programas, a abordagem utiliza técnicas estatísticas e síntese de programas baseada em tipos. Inicialmente, a abordagem utiliza estatística para identificar as localizações que podem conter faltas e atribui uma pontuação para cada localização. Baseado nessas localizações, a técnica usa um algoritmo de pesquisa para sintetizar programas que substituem as localizações com faltas com localizações corretas.

Por fim, o trabalho P21 realiza uma análise sobre os aspectos de similaridade entre a linguagem natural (i.e., texto de um idioma) e a linguagem de programação. Baseado nos aspectos similares entre o texto e uma linguagem de programação. O trabalho aborda características que podem ser empregadas para que um usuário aumente o seu conhecimento sobre uma linguagem de programação, sem a presença de um tutor. Para fazer essa análise, o trabalho propõe sistemas automatizados para mapear o conhecimento em um determinado domínio da linguagem de programação para a mineração de repositórios de software.

3.8. Considerações Finais do Capítulo

Esse capítulo apresentou uma revisão sistemática a respeito das ferramentas de detecção e correção de erros de sintaxe por meio de processamento de linguagem natural. Tendo como resultado 21 trabalhos válidos, que se enquadram nos critérios de exclusão e inclusão, e com isso respondem as questões de pesquisa apresentadas (Seção 3.2). Além disso, apresenta uma breve descrição de todas as ferramentas resultantes na pesquisa.

As respostas a questão de pesquisas RQ1 é visualizada através do Gráfico 3 que apresentam as abordagens de aprendizado de máquina, resultantes da pesquisa, que auxiliam os estudantes na resolução de tarefas de programação. Já a resposta a questão de pesquisa RQ2 é visualizada por meio do Gráfico 5 que apresenta quais os erros que alvos das abordagens resultantes da pesquisa.

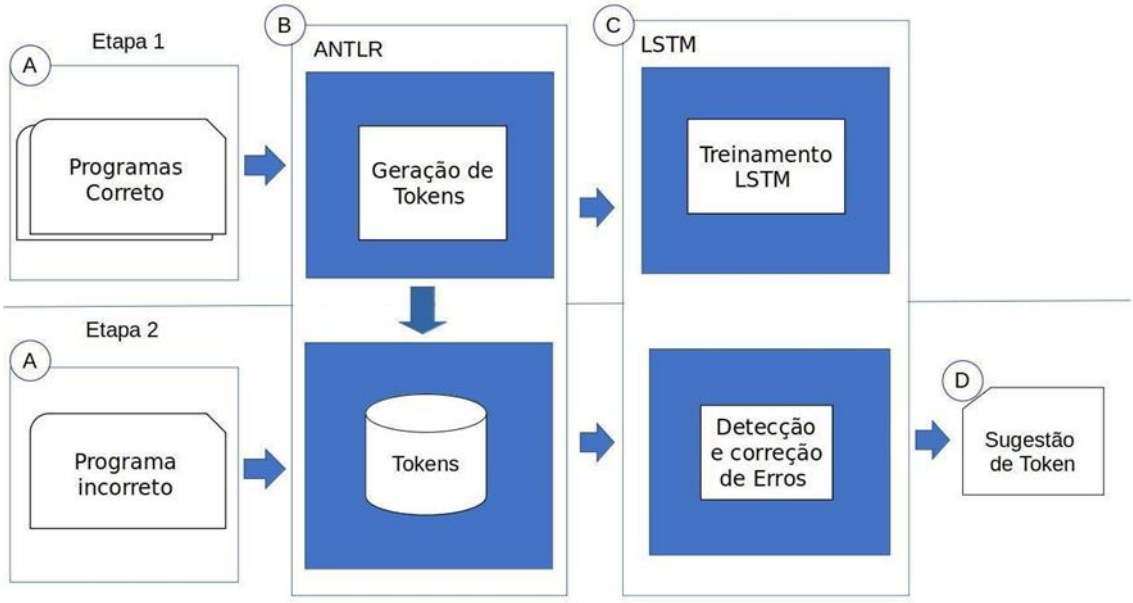
Dentre as ferramentas destacam-se as apresentadas pelos trabalhos P1 e P2, aos quais utilizam RNN para localizar e corrigir erros sintáticos e de compilação gerados por alunos novatos e que utilizam (MOOC). Tais características contribuem de maneira direta para

elaboração da ferramenta proposta por esse trabalho, pois estão voltadas a auxiliar alunos de programação na resolução de erros sintáticos durante a realização de tarefas e utilizam processamento de linguagem natural.

4. UMA ABORDAGEM PARA DETECÇÃO E CORREÇÃO DE ERROS SINTÁTICOS

Neste capítulo descreve-se a ferramenta proposta para detecção e correção de erros. A ferramenta foi desenvolvida utilizando a linguagem de programação Python por meio da plataforma de interface gráfica do usuário Anaconda *Navigator*[2] que permite realizar a inicialização de programas e gerenciar de maneira prática pacotes, ambientes e canais conda sem realizar a utilização de comando da linha de comando. Antes de apresentar os detalhes da ferramenta, apresenta-se uma visão geral da mesma. A ferramenta apresenta duas etapas base. A primeira, é a etapa de treinamento, que recebe como entrada os programas bases para o aprendizado da LSTM; e a etapa de localização e correção, que recebe como entrada o programa incorreto de um estudante a ser analisado pela ferramenta. A Figura 4 apresenta as duas etapas seguidas pela ferramenta desenvolvida. Na primeira etapa, a ferramenta possui como entrada programas corretos desenvolvidos em linguagem C (Figura 4A).

Figura 4: Estrutura Sequencial da Abordagem



Fonte: Próprio Autor

O propósito dos programas de entrada é servir como modelo de programas a serem desenvolvidos, que foram escritos por programadores com certa experiência (por exemplo, disponíveis em repositórios de software no GitHub[3] de competidores de maratonas de

[2] https://docs.anaconda.com/anaconda/navigator/
[3] https://github.com/

programação) e não apresenta erros sintáticos. Esses dados na abordagem proposta são denominados de dados de treino. Dessa forma, qualquer programa que se desvia da construção sintática desses programas pode ser considerado incorretos. Um exemplo de um programa sintaticamente incorreto na linguagem C é aquele que não possui um número balanceado de chaves de abertura e fechamento dos elementos da linguagem de programação analisada. Porém, esses programas de entrada estão escritos em texto puro. Dessa forma, para se realizar uma análise dos mesmos, é necessário convertê-los para elementos da linguagem de programação. Uma forma de se alcançar isso é usar processos comum na construção de compiladores tais como a análise léxica e sintática. Dessa forma, os programas planos escritos em texto, são submetidos ao analisador gramatical ANTLR, que detecta cada elemento gramatical da linguagem presentes nos programas passados como entradas, e os convertem em *tokens* (Figura 4 B), ou seja, realiza uma análise léxica dos programas passados como entrada. O algoritmo de aprendizagem de máquina utilizado, o LSTM, recebe a sequência de todos para todos os programas de entrada. Esses dados são utilizados para treinar a rede neural LSTM (Figura 4 C).

Uma vez que a rede neural foi treinada, ela pode ser utilizada para detectar problemas em programas incorretos dos estudantes e sugerir correções, o que é feito na segunda etapa da ferramenta. Na segunda etapa, a ferramenta recebe como entrada programas desenvolvidos em linguagem C que apresentam possíveis erros sintáticos (Figura 4 A). A ideia é analisar esse programa, identificar os possíveis erros sintáticos e sugerir formas de corrigi-lo. Dada mesma forma que na primeira etapa da ferramenta, esses problemas estão escritos em texto plano. Dessa forma, eles precisam ser convertidos para elementos da linguagem de programação. O programa (possivelmente incorreto) é processado pelo analisador ANLR e seus *tokens* são gerados (Figura 4 B). Em seguida, a ferramenta utiliza a rede neural LSTM treinada para localizar e sugerir formas de corrigir o programa fornecido como entrada (Figura 4 C). Por fim, a ferramenta apresenta ao usuário uma possível sugestão da resolução do erro (Figura 4 D). As correções de erros propostas são do tipo, de identificar que tipo de *token* está posicionado incorretamente e sugerir qual *token* deveria estar naquela localização. Dessa forma, a ferramenta proposta sugere um único *token* e não uma sequência de *tokens* como sugestão.

4.1. Programas de Entrada

Uma vez apresentada uma visão geral da ferramenta, nesta seção detalha-se os programas que são utilizados por ela como entrada para treinar a rede neural LSTM e o programa incorreto do qual os erros sintáticos e sugestões são identificados. Os programas de entrada utilizados pela ferramenta são ambos desenvolvidos na linguagem de programação C. A linguagem de programação foi escolhida para ser analisada devido sua grande utilização nas disciplinas introdutórias de programação ofertadas por instituições de ensino superior. No entanto, como a abordagem utiliza um analisador de linguagem (o ANTLR) que é independente da linguagem de programação sob análise, a ferramenta poderia ser utilizada para resolver problemas sintáticos em qualquer linguagem de programação, desde de que a gramática da mesma esteja disponível para uso no ANTLR. Os *tokens* dos programas corretos possuem a finalidade de serem objetos de treinamentos da LSTM para que a ferramenta possa utilizar essa rede neural para realizar as possíveis correções de programas incorretos na segunda etapa da ferramenta (seção anterior).

Os programas corretos utilizados para treinar a rede neural LSTM foram selecionados de repositórios abertos do GitHub devido sua popularização entre estudantes de programação e programadores profissionais decorrente das diversas funcionalidades que o GitHub disponibiliza para facilitar e organizar o processo de desenvolvimento de programas. Todos os programas corretos são referentes a resoluções de problemas disponibilizados pela plataforma URI[4].

A Figura 5 apresenta um exemplo simples de código correto desenvolvido utilizando a linguagem de programação C que foi obtido dos repositórios do GitHub, o mesmo é a resolução de uma das tarefas disponibilizadas pela plataforma URI. O programa solicita as 3 notas do usuário e realiza o cálculo da média e apresenta ao usuário.

[4] https://www.urionlinejudge.com.br/judge/pt/login?redirect=%2Fpt

Figura 5: Código Correto em C

```c
#include <stdio.h>

int main()
{

    float A, B, C;
    double MEDIA;

    A = B = C = 10;

    scanf("%f", &A);
    scanf("%f", &B);
    scanf("%f", &C);

    MEDIA = ((2*A) + (3*B) + (5*C)) / 10;

    printf("MEDIA = %.1lf\n", MEDIA);

    return (0);
}
```

Fonte**:** Próprio Autor.

Na segunda etapa, a ferramenta recebe como entrada um programa incorreto para identificar e gerar sugestões. Dessa forma, o programa incorreto inserido com entrada na ferramenta, tem a finalidade de ser processado e avaliado pela mesma, para que possa ser proposto uma correção referente a possíveis erros sintáticos. A identificação e correção de erros sintáticos foram escolhidos como foco deste trabalho, pois os mesmos são um dos mais comuns realizados por programadores iniciantes e que possui difícil interpretação quando apresentados por meio de mensagem por algumas IDE's, que até mesmo em alguns casos, apresenta mensagens errôneas.

Figura 6: Programa com erros de sintaxe

```c
#include <stdio.h>

int main()
        int a = 10;
        int b= 3
        int a = a - b
        printf(" resultado = %s ", a);
        return 0;
}
```

Fonte: Próprio autor.

A Figura 6 apresenta o código de um programa escrito utilizando a linguagem de programação C contendo erros sintáticos, como observado nas linhas 3, 5, 6 e 7 tais erros são

referentes a não inserção do abre chaves (linha 3) ausência do ponto e vírgula (linhas 5 e 6) e da inserção do “%s” ao invés do “%d” ou “%i” (linha 7). Dos erros, apresentados na Figura 6, a ferramenta ajuda na resolução de erros sintáticos, o erro semântico relacionado ao uso do %s ao invés do %d não é foco da ferramenta proposta.

Figura 7: Programa com correção de erros sintáticos

```
1 #include <stdio.h>
2
3 int main(){
4         int a = 10;
5         int b= 3;
6         int a = a - b;
7         printf(" resultado = %d ", a);
8         return 0;
9 }
```

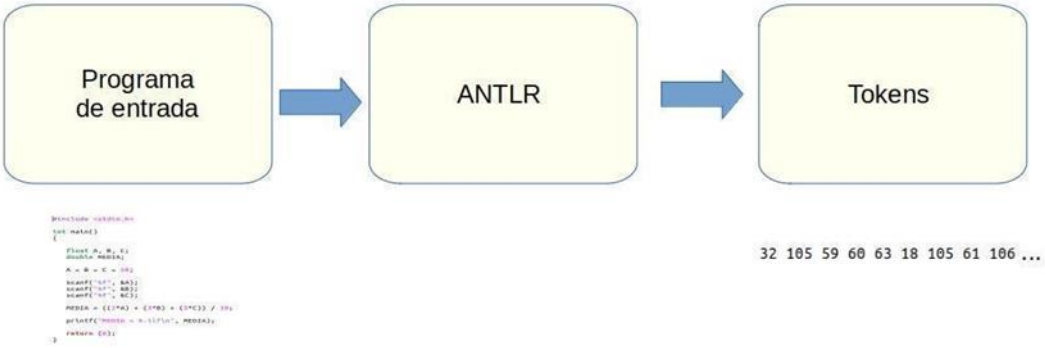
Fonte: Próprio autor.

Após o código incorreto ser processado e avaliado, a ferramenta dispõe ao usuário uma proposta de possíveis correções dos erros sintáticos, para que o estudante possa realizar as devidas alterações. A Figura 7 apresenta correções realizadas pelo usuário mediante as sugestões propostas pela ferramenta. As correções são referentes aos erros apresentados pela Figura 6, em que é inserido o abre chaves (linha 3), e do ponto e vírgula (linha 5 e 6).

4.2. Geração de *Tokens*

Como discutido anteriormente, ao inserir os códigos corretos dos programas na ferramenta, antes que o treinamento da LSTM ocorra, necessita-se da etapa de geração de *tokens*, uma vez que os programas de entrada estão escritos em texto plano. Neste processo, o ANTLR é utilizado para processar, identificar e realizar a tradução de cada elemento gramatical da linguagem C em *tokens*. Para que isso seja possível, durante o processo de instalação do ANTLR, informa-se um arquivo referente a estrutura da gramática C. A partir de então, o ANTLR transforma o arquivo gramatical em duas partes, são elas: o CLexer, que realiza a leitura dos arquivos de entradas e os transforma em *tokens*; e o CParser, que representa o *parser* (analisador sintático) da linguagem. Apesar do ANTLR gerar os dois tipos de analisadores, o léxico e o sintático, no trabalho apenas o analisador léxico foi utilizado, uma vez que a ferramenta apenas identifica erros sintáticos da linguagem de programação.

Figura 8: Sequência de geração de *tokens*



Fonte: Próprio Autor (2020).

A Figura 8 apresenta os passos necessários para a geração de *tokens*, que consiste em passar os programas corretos, que são inseridos como entrada na ferramenta, para um processamento realizado pelo analisador léxico gerado pelo ANTLR e resultando como saída, os *tokens* referentes aos elementos gramaticais dos programas passados como entrada. A cada programa de treinamento a ser processado, o processo acima é realizado, gerando como saída uma sequência de *tokens* presentes para todos os programas no conjunto de treinamento.

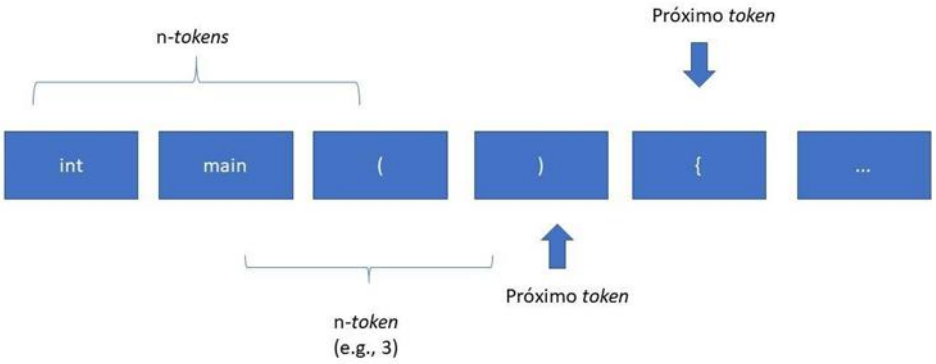
4.3. Treinamento da LSTM

Nesta seção descreve-se o treinamento da rede neural LSTM. Para que o treinamento da LSTM seja efetivado, precisa-se que os *tokens* gerados na etapa anterior (Seção 4.2) sejam processados pela LSTM, uma vez que esses *tokens* são a entrada desta rede neural na abordagem proposta.

Usando essa sequência de *tokens* como entrada, no processo de treinamento da rede neural n *tokens*, n *tokens* são selecionados em uma janela de processamento e sugere como saída o próximo *token*. Por exemplo, a sequência de *tokens* corresponde ao trecho de código “int main () {...” é mostrada na Figura 9. Se a janela de processamento for de 3 *tokens*, então três *tokens* serão selecionados e como saída, a rede neural sugere como saída o próximo *token*. Se o valor *token* sugerido pela rede neural for diferente do próximo *token*, os pesos da rede neural devem ser atualizados de forma a considerar que nesse caso a resposta foi incorreta. O treinamento

segue o processo e seleciona os próximos 3 *tokens* e sugere o próximo *token*, que é novamente analisado para refinar a rede. Esse processo continua até que não haja mais *tokens* a serem considerados.

Figura 9: Ilustração do processo de treinamento da rede neural



Fonte: Próprio autor.

Neste trabalho, a saída da rede neural é um dos n *tokens* da linguagem. Dessa forma, múltiplas saídas são possíveis. Por exemplo, o quão provável é que o próximo token seja um "int", o quão provável que seja um "(", o quão provável que seja um "{" e assim por diante. Para permitir que múltiplas classes sejam sugeridas, é preciso gerar como saída um vetor "*one hot*". Um vetor "*one hot*" é um vetor que possui 1 em um de seus índices e 0 em todos os outros índices. Para que a sugestão de *tokens* seja possível na ferramenta proposta, foi construído um mapa chave e valor em que cada token da linguagem pertence a um índice que o identifica (Tabela 6). Dessa forma, a rede neural LSTM gera como saída um vetor "*one hot*" com um valor que identifica o *token* de saída. Por exemplo, se a rede neural gerar um vetor que possui um 1 na posição 0 do vetor, indica que o *token* sugerido é o comando "*Break*" de acordo com a Tabela 6.

Tabela 6: Mapeamento dos tokens da linguagem para índice

Break	**0**
Case	1
Char	2
Const	3
Continue	4
...	...

Fonte: Próprio Autor

Para facilitar a identificação do mapeamento do *token* para o índice e do índice para o *token* foram criados dois mapas. Um mapa contendo *tokens* como chave e os índices do vetor

como valor e um mapa reverso que possui a finalidade de realizar a decodificação de cada elementos de saída da LSTM.

A partir de então, define-se os parâmetros de treinamento da rede, são eles: o número de unidades contidas na célula RNN obtendo o valor de 512; a taxa de aprendizagem (*learning rate*) com o valor de 1%; número de itens por iterações com o valor de 1000 elementos; e por fim define-se uma quantidade de 3 elementos de entradas na rede.

Para obter o valor 0 ou 1 utilizado no vetor *one hot* utiliza-se uma função *logit*, que retorna o valor de probabilidade de uma dada classificação de *tokens*, correspondente a cada token passado como entrada para a função, restringindo assim um valor n-*tokens* de uma grande escala para dentro do intervalo de 0 a 1.

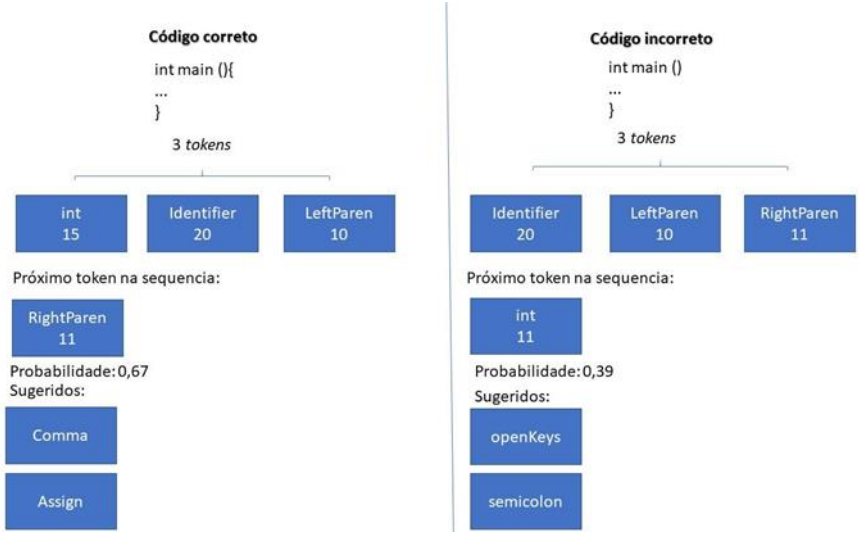
4.4. Detecção e Correção de Erros Sintáticos

Para a detecção e a correção de problemas sintáticos no código, utilizou-se uma abordagem de escanear o código incorreto submetido pelo estudante. Para isso, utilizamos uma janela de *tokens*. Iniciando pelo início do programa incorreto, seleciona-se n *tokens* da esquerda para a direita. Fornecendo esses n *tokens* como entrada da rede neural LSTM, ela produz como saída a probabilidade de cada um dos *tokens* da gramática serem o próximo *token* da que os n *tokens* anteriores. Em seguida, verifica-se o próximo *token* que o programa incorreto apresenta após os n *tokens* de entrada. Se a probabilidade do próximo *token* do programa incorreto for muito pequena, isso é um indicativo de que o próximo *token* do programa está incorreto. Ou seja, os programas da base de dados, escritos por programadores experientes podem não apresentar a sequência dos n + 1 *tokens* (n *tokens* + o próximo *token*). Quando isso ocorre, a ferramenta sugere um conjunto de k *tokens* que poderiam substituir o *token* problemático. Esse conjunto de n *tokens* sugeridos são ordenados de forma decrescente de acordo com a sua probabilidade de ocorrência, ou seja, quanto mais fácil é para o *token* aparecer, maior será as suas chances de serem sugeridos.

Esta abordagem para sugerir problemas sintáticos também foi empregada no trabalho de Santos et al. (2018b). A Figura 10 ilustra em seu lado esquerdo os *tokens* de um trecho de código que apresenta ausência de erros sintáticos, tal constatação se dá mediante que o *token* 11 apresentar elevada probabilidade de ocorrência (0.67). De outra forma, o lado direito da

Figura 10 apresenta-se os *tokens* de um trecho de código incorreto, com a probabilidade de 0.39 do *token* 11 ser o candidato a token no referido local do código.

Figura 10: Exemplo de detecção de problema sintático



Fonte: Próprio Autor.

Para efetivação do treinamento da ferramenta utilizou-se 67 programas desenvolvidos utilizando a linguagem de programação C. Após o programa concluir o treinamento e apresentar uma acurácia superior a 50%.

4.5. Considerações Finais da Abordagem Apresentada

Este capítulo apresentou uma ferramenta que utiliza processamento de linguagem (LSTM) para auxiliar estudantes na resolução de tarefas de programação, no tocante a localização e correção de erros sintáticos. A ferramenta desenvolvida utiliza a linguagem de programação Python e apresentou resultados satisfatórios no tocante a seus objetivos, proporcionando correções precisa, bem como, suas características satisfazem as questões de pesquisa apresentadas na Seção 3.3, como uma ferramenta que utiliza processamento de linguagem natural que dispõem suporte aos alunos no processo de resolução de tarefas de programação, no tocante a identificação e correção de erros sintáticos.

5. CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foi proposta uma ferramenta para correção de erros sintáticos usando aprendizado de máquina. Ao resolver problemas de programação os estudantes podem conter dificuldade com relação a corrigir problemas sintáticos do programa que está sendo construído. Apesar de os ambientes de desenvolvimento integrados fornecerem sugestões sobre como corrigir o programa. As mensagens de erro exibidas pelos compiladores por vezes não ajudam um programador iniciante a resolver o problema sintático, podendo a linha marcada como problemática se apresentar distante da linha fornecida pelo compilador. Para a construção da ferramenta, primeiramente foi realizada uma revisão sistemática da literatura sobre as técnicas de aprendizagem de máquina que foram utilizadas para auxiliar estudantes na resolução de problemas de programação. Após a análise da literatura foi desenvolvida uma ferramenta para correção automática de problemas sintáticos. A ferramenta recebe como entrada um programa incorreto e fornece sugestões de que *tokens* da linguagem poderia ser utilizado para resolver um problema sintático do estudante.

Como trabalhos futuros, pretende-se criar um ambiente Web para permitir que o discente interaja com a ferramenta apresentada por esse trabalho, bem como, aprimorar o método de correção para que seja possível localizar e corrigir erros semânticos.

REFERÊNCIAS

AHO, A. V.; LAM, M. S.; SETHI, R.; ULLMAN, J. D. **Compiladores: Princípios, técnicas e ferramentas**. 2ª Edição, São Paulo: Pearson Addison-Wesley. 2008. P. 248.

BECKER, Brett A. An Effective Approach to Enhancing Compiler Error Messages. **THE 47TH ACM TECHNICAL SYMPOSIUM,** [S.l.]: ACM Press, 2016. Disponível em: <http://dx.doi.org/10.1145/2839509.2844584>.

BECKER, Brett A. An Exploration of the Effects of Enhanced Compiler Error Messages for Computer Programming Novices. . [S.l.]: Unpublished. 2015. Disponível em: <http://rgdoi.net/10.13140/RG.2.2.26637.13288>.

BHATIA, Sahil; SINGH, Rishabh. Automated correction for syntax errors in programming assignments using recurrent neural networks**. arXiv preprint arXiv:1603.06129**, 2016.

BRAGA, A. P.; CARVALHO, A. P. L. F.; LUDEMIR, T. B. Fundamentos de redes neurais artificiais. Fundamentos de redes neurais artificiais. **Rio de Janeiro: 11a Escola de Computação**, 1998.

BROWN, Neil CC; ALTADMRI, Amjad. Investigating novice programming mistakes: educator beliefs vs. student data. **Proceedings of the tenth annual conference on International computing education research**. ACM, 2014. p. 43-50.

CAMPBELL, Joshua Charles; HINDLE, Abram; AMARAL, José Nelson. Syntax errors just aren't natural: improving error reporting with language models. **THE 11TH WORKING CONFERENCE**, [S.l.]: ACM Press, 2014. Disponível em: <http://dx.doi.org/10.1145/2597073.2597102>.

DUH, Mei-Sheng; WALKER, Alexander M; PAGANO, Marcello; KRONLUND, Kenneth. Prediction and Cross-Validation of Neural Networks Versus Logistic Regression: Using Hepatic Disorders as an Example**. American Journal of Epidemiology**, v. 147, n. 4, p. 407–413, 15 fev. 1998. Disponível em: <http://dx.doi.org/10.1093/oxfordjournals.aje.a009464>.

EL-SOLH, A. A.; HSIAO, C.-B.; GOODNOUGH, S.; SERGHANI, J.; GRANT, B. J. B. Predicting Active Pulmonary Tuberculosis Using an Artificial Neural Network. **Chest**, v. 116, n. 4, p. 968–973, out. 1999. Disponível em: <http://dx.doi.org/10.1378/chest.116.4.968>.

GANSSLE, G. Neural networks. The Leading Edge, [s. l.], v. 37, n. 8, p. 616–619, 2018. Disponível em: <http://dx.doi.org/10.1190/tle37080616.1>.

GANSSLE, Graham. Neural networks. The Leading Edge. https://doi-org.ez13.periodicos.capes.gov.br/10.1190/tle37080616.1. 2018.

GERS, F.A. Learning to forget: continual prediction with LSTM. **9th International Conference On Artificial Neural Networks (ICANN)** , 1999, [S.l.]: IEE, 1999. Disponível em: <http://dx.doi.org/10.1049/cp:19991218>.

GERS, F.A.; SCHMIDHUBER, E. LSTM recurrent networks learn simple context-free and context-sensitive languages. **IEEE Transactions on Neural Networks**, v. 12, n. 6, p. 1333–1340, 2001. Disponível em: <http://dx.doi.org/10.1109/72.963769>.

GERS, Felix A.; SCHRAUDOLPH, Nicol N.; SCHMIDHUBER, Jürgen. Learning precise timing with LSTM recurrent networks. **Journal of Machine Learning Research**, v. 3, n. Aug, p. 115-143, 2002.

GUPTA, Rahul; Pal, Soham; Kanade, Aditya; Shevade, Shirish . Deepfix: Fixing common c language errors by deep learning. **Thirty-First AAAI Conference on Artificial Intelligence**. 2017.

GOMES, Anabela de Jesus. **Dificuldades de aprendizagem de programação de computadores: contributos para a sua compreensão e resolução**. Tese de Doutorado. 2010.

GORGENS, Eric Bastos; LEITE, Helio Garcia; SANTOS, Heleno do Nascimento; GLERIANI, José Marinaldo. Estimação do volume de árvores utilizando redes neurais artificiais. **Revista Árvore**, v. 33, n. 6, p. 1141-1147, 2009.

GREFF, K.; SRIVASTAVA, R. K.; KOUTNIK, J.; STEUNEBRINK, B. R.; SCHMIDHUBER, J. LSTM: A Search Space Odyssey**. IEEE Transactions on Neural Networks and Learning Systems**, v. 28, n. 10, p. 2222–2232, out. 2017. Disponível em: <http://dx.doi.org/10.1109/TNNLS.2016.2582924>.

HAMMAD, T. A.; ABDEL-WAHAB, M. F.; DECLARIS, N.; EL-SAHLY, A.; EL-KADY, N.; STRICKLAND, G. T. Comparative Evaluation of the Use of Artificial Neural Networks for Modelling the Epidemiology of Schistosomiasis Mansoni. **Transactions of the Royal Society of Tropical Medicine and Hygiene**, v. 90, n. 4, p. 372–376, jul. 1996. Disponível em: <http://dx.doi.org/10.1016/s0035-9203(96)90509-x>.

HAYKIN, S. **Neural Networks: A comparative Foundation**. New Jersey. Prentice Hall. 1999.

HELLENDOORN, Vincent J.; DEVANBU, Premkumar. Are deep neural networks the best choice for modeling source code?. **Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering**. ACM, 2017. p. 763-773.

JÚNIOR, J. C. R. P. et al. Ensino de algoritmos e programação: uma experiência no nível médio. In: **XIII Workshop de Educação em Computação (WEI'2005). São Leopoldo, RS, Brasil**. 2005

KOVACS, Z. L. **Redes neurais artificiais**: fundamentos e aplicações. 2.ed. São Paulo: Colledium Cognitio, 1996. 174p.

LAPEER, R. J. A.; DALTON, K. J.; PRAGER, R. W.; FORSSTRÖM, J. J.; SELBMANN, H. K.; DEROM, R. Application of Neural Networks to the Ranking of Perinatal Variables Influencing Birthweight. **Scandinavian Journal of Clinical and Laboratory Investigation**, v. 55, n. sup222, p. 83–93, jan. 1995. Disponível em: <http://dx.doi.org/10.3109/00365519509088454>.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep learning. **Nature**, v. 521, n. 7553, p. 436, 2015.

MOREIRA, Gabriel Luídy; HOLANDA, Wallace; COUTINHO, Jarbele Cássia da S.; CHAGAS, Ferdinandy S. Desafios na aprendizagem de programação introdutória em cursos de TI da UFERSA, campus Pau dos Ferros: um estudo exploratório. **Anais do Encontro de Computação do Oeste Potiguar ECOP/UFERSA**, v. 2, n. 1, 2018.

NELSON, David Michel Quirino. Uso de redes neurais recorrentes para previsão de séries temporais financeiras. 2017.

PARR, T.; FISHER, K. LL(*): The Foundation of the ANTLR Parser Generator. **ACM SIGPLAN Notices**, [s. l.], v. 46, n. 6, p. 425, 2011. Disponível em: <http://dx.doi.org/10.1145/1993316.1993548>.

PARR, Terence.T**he definitive ANTLR 4 reference**. Pragmatic Bookshelf, 2013.

RAUBER, Thomas Walter. **Redes neurais artificiais**. Universidade Federal do Espírito Santo: Departamento de informática, Vitória. 2005.

RAYCHEV, Veselin; VECHEV, Martin; YAHAV, Eran. Code completion with statistical language models. **Acm Sigplan Notices**. ACM, 2014. p. 419-428.

RIPLEY, R. M.; HARRIS, A. L.; TARASSENKO, L. Neural Network Models for Breast Cancer Prognosis. **Neural Computing & Applications**, v. 7, n. 4, p. 367–375, dez. 1998. Disponível em: <http://dx.doi.org/10.1007/BF01428127>.

SAK, Haşim; SENIOR, Andrew; BEAUFAYS, Françoise. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. **Proceedings of the Annual Conference of the International Speech Communication Association, (INTERSPEECH).** P. 338-342. 2014.

SANTOS, Alcione Miranda dos; SEIXAS, José Manoel de; PEREIRA, Basílio de Bragança; MEDRONHO, Roberto de Andrade. Usando redes neurais artificiais e regressão logística na predição da hepatite A**. Revista Brasileira de Epidemiologia**, v. 8, p. 117-126, 2005. Disponível em: <http://dx.doi.org/10.1590/S1415-790X2005000200004>.

SANTOS, E. A.; CAMPBELL, J. C.; PATEL, D.; HINDLE, A.; AMARAL, J. N. Syntax and sensibility: Using language models to detect and correct syntax errors. **2018 Ieee 25th International Conference On Software Analysis, Evolution And Reengineering (SANER)**, mar. 2018, [S.l.]: IEEE, mar. 2018b. Disponível em: <http://dx.doi.org/10.1109/SANER.2018.8330219>.

SANTOS, Priscila SC; ARAUJO, Luis Gustavo J.; BITTENCOURT, Roberto A. A Mapping Study of Computational Thinking and Programming in Brazilian K-12 Education. **2018 IEEE Frontiers in Education Conference (FIE)**. IEEE, 2018a. p. 1-8.

SBC - Brazilian Computer Society**, Referenciais de Formação em Computação: EducaçãoBásica**, Relatório Tecnico. 2017.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação**. 11ª Edição, Bookman Editora, 2018.

SILVA, Salvador Ramos Bernardino da. **Software adaptativo: método de projeto, representação gráfica e implementação de linguagem de programação**. Dissertação de Mestrado - Universidade de São Paulo. São Paulo 2011.

SOMMERVILLE, Ian. **Engenharia de Software**, ed. 9ª. São Palo, SP, Brasil, 2011.

SOUSA, Reudismam Rolim de; LEITE, Felipe Torres; GUIMARÃES, Ádller de Oliveira; OLIVEIRA, Assunaueny Rodrigues de. Pré-algoritmos – Ações de apoio à melhoria do ensino de graduação. **IV Encontro de Computação do Oeste Potiguar (ECOP)**. Pau dos Ferros. 20119.

TABANAO, Emily S.; RODRIGO, Ma Mercedes T.; JADUD, Matthew C. Identifying at-risk novice java programmers through the analysis of online protocols. **Philippine Computing Science Congress**. 2008. p. 1-8.

WHITE, Martin et al. Toward deep learning software repositories. **Proceedings of the 12th Working Conference on Mining Software Repositories**. IEEE Press, p. 334-345, 2015.

APÊNDICE

APÊNDICE A – *STRING* DE PESQUISA PARA CADA BASE DE DADOS

ACM

content.ftsec:(+("Recurrent neural networks" "Artificial Intelligence" "Machine Learning" "Deep learning" "Natural language processing")+(Programmer student developer) +("Program repair" "Programming education" "Syntactic Errors"))

IEEE

((("Full Text & Metadata":"Recurrent neural networks" OR "Artificial Intelligence" OR "Machine Learning" OR "Deep learning" OR "Natural language processing") AND "Full Text & Metadata":"Programmer" OR "student" OR "developer") AND "Full Text & Metadata":"Program repair" OR "Programming education" OR "Syntactic Errors")

Scopus

(ALL ("Recurrent neural networks" OR "Artificial Intelligence" OR "Machine Learning" OR "Deep learning" OR "Natural language processing") AND ALL (programmer OR student OR developer) AND ALL ("Program repair" OR "Programming education" OR "Syntactic Errors")) AND PUBYEAR > 2014

APÊNDICE B – LISTA DE REFERÊNCIAS DOS TRABALHOS INCLUSOS

[P1] AHMED, Umair Z.; KUMAR, Pawan; KARKARE, Amey. Compilation error repair: For the Student Programs, From the Student Programs. **THE 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)**, [S.l.]: ACM Press, 2018. Disponível em: <http://dx.doi.org/10.1145/3183377.3183383>.

[P2] BHATIA, Sahil; KOHLI, Pushmeet; SINGH, Rishabh. Neuro-symbolic program corrector for introductory programming assignments. **THE 40th International Conference on Software Engineering (ICSE)**, [S.l.]: ACM Press, 2018. Disponível em: <http://dx.doi.org/10.1145/3180155.3180219>.

[P3] YOSHIZAWA, Yuto; WATANOBE, Yutaka. Logic Error Detection Algorithm for Novice Programmers based on Structure Pattern and Error Degree. **The 9TH INTERNATIONAL CONFERENCE ON AWARENESS SCIENCE AND TECHNOLOGY (ICAST)**, [S.l.]: IEEE, set. 2018. Disponível em: <http://dx.doi.org/10.1109/ICAwST.2018.8517171>.

[P4] BIRCH, Geoff; FISCHER, Bernd; POPPLETON, Michael. Fast Test Suite-Driven Model-Based Fault Localisation with Application to Pinpointing Defects in Student Programs. **Software & Systems Modeling (SoSyM)**, v. 18, n. 1, p. 445–471, 27 jul. 2017. Disponível em: <http://dx.doi.org/10.1007/s10270-017-0612-y>.

[P5] SILVA, Priscylla; COSTA, Evandro; DE ARAÚJO, Joseana Régis. An Adaptive Approach to Provide Feedback for Students in Programming Problem Solving. **Intelligent Tutoring Systems (ITS)**. [S.l.]: Springer International Publishing, p. 14–23. 2019. Disponível em: <http://dx.doi.org/10.1007/978-3-030-22244-4_3>.

[P6] WANG, Ke; SINGH, Rishabh; SU, Zhendong. Search, Align, and Repair: Data-Driven Feedback Generation for Introductory Programming Exercises. **The 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)**, v. 53, n. 4, p. 481–495, 11 jun. 2018. Disponível em: <http://dx.doi.org/10.1145/3296979.3192384>.

[P7] SHARMA, Saksham; AGARWAL, Pallav; MOR, Parv; KARKARE, Amey . TipsC: Tips and Corrections for programming MOOCs. **Artificial Intelligence in Education (AIED).** [S.l.]: Springer International Publishing, 2018. p. 322–326. Disponível em: <http://dx.doi.org/10.1007/978-3-319-93846-2_60>.

[P8] EICHER, Bobbie Lynn; JOYNER, David; GOEL, Ashok. Leveraging Mutual Theory of Mind for More Human-Like Intelligent Tutoring Systems. **Springer International Publishing AG**. 2018. Disponível em: <https://doi.org/10.1007/978-3-319-91464-0>.

[P9] BUI, Hieu Trong; F D SYED MUSTAPHA, Syed Malek. Automated Data-Driven Hint Generation in Intelligent Tutoring Systems for Code-Writing: On the Road of Future Research**. International Journal of Emerging Technologies in Learning (iJET)**, v. 13, n. 9, p. 174, 29 set. 2018. Disponível em: <http://dx.doi.org/10.3991/ijet.v13i09.8023>.

[P10] KATO, Toshiyasu et al. Analysis of Students' Behaviors in Programming Exercises Using Deep Learning. **Smart Education and e-Learning 2017 (SEEL)**. [S.l.]: Springer International Publishing, 2017. p. 38–47. Disponível em: <http://dx.doi.org/10.1007/978-3-319-59451-4_4>.

[P11] KIM, D.; KWON, Y.; LIU, P.; KIM, I. L.; PERRY, D. M.; ZHANG, X.; RODRIGUEZ-RIVERA. Apex: automatic programming assignment error explanation. **The 2016 ACM SIGPLAN International Conference On Object-Oriented Programming, Systems, Languages, And Applications (OOPSLA),** [S.l.]: ACM Press, 2016. Disponível em: <http://dx.doi.org/10.1145/2983990.2984031>.

[P12] BIRCH, Geoff; FISCHER, Bernd; POPPLETON, Michael. Using Fast Model-Based Fault Localisation to Aid Students in Self-Guided Program Repair and to Improve Assessment. **The 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)**, [S.l.]: ACM Press, 2016. Disponível em: <http://dx.doi.org/10.1145/2899415.2899433>.

[P13] YI, Jooyong; AHMED, Umair Z; KARKARE, Amey; TAM, Shin Hwei; ROYCHOUDHURY, Abhik. A feasibility study of using automated program repair for introductory programming assignments. T**he 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE),** [S.l.]: ACM Press, 2017. Disponível em: <http://dx.doi.org/10.1145/3106237.3106262>.

[P14] PU, Y.; NARASIMHAN, K.; SOLAR-LEZAMA, A.; BARZILAY, R. sk_p: a neural program corrector for MOOCs. **Companion Proceedings Of The 2016 ACM Sigplan International Conference On Systems, Programming, Languages And Applications: Software For Humanity (Splash)**, Anais.... In: COMPANION THE 2016 ACM SIGPLAN

INTERNATIONAL CONFERENCE.: ACM Press, 2016. Disponível em: <http://dx.doi.org/10.1145/2984043.2989222>.

[P15] AHADI, Alireza. Early Identification of Novice Programmers' Challenges in Coding Using Machine Learning Techniques. **Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER)**, [S.l.]: ACM Press, 2016. Disponível em: <http://dx.doi.org/10.1145/2960310.2960339>.

[P16] GULWANI, Sumit; RADIČEK, Ivan; ZULEGER, Florian. Automated Clustering and Program Repair for Introductory Programming Assignments. T**he 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)**, v. 53, n. 4, p. 465–480, 11 jun. 2018. Disponível em: <http://dx.doi.org/10.1145/3296979.3192387>.

[P17] SONG, Dowon; LEE, Myungho; OH, Hakjoo. Automatic and Scalable Detection of Logical Errors in Functional Programming Assignments**. Proceedings of the ACM on Programming Languages**, v. 3, n. OOPSLA, p. 1–30, 10 out. 2019. Disponível em: <http://dx.doi.org/10.1145/3360614>.

[P18] AHADI, Alireza; LISTER, Raymond; HAAPALA, Heikki; VIHAVAINEN, Arto. Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance. **Proceedings of the eleventh annual International Conference on International Computing Education Research (ICER)**, [S.l.]: ACM Press, 2015. Disponível em: <http://dx.doi.org/10.1145/2787622.2787717>.

[P19] AHADI, Alireza; LISTER, Raymond; MATHIESON, Luke. Syntax error based quantification of the learning progress of the novice programmer. **The 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)**, [S.l.]: ACM Press, 2018. Disponível em: <http://dx.doi.org/10.1145/3197091.3197121>.

[P20] LEE, Junho; SONG, Dowon; SO, Sunbeom; OH, Hakjoo. Automatic Diagnosis and Correction of Logical Errors for Functional Programming Assignments. **Proceedings of the ACM on Programming Languages**, v. 2, n. OOPSLA, p. 1–30, 24 out. 2018. Disponível em: <http://dx.doi.org/10.1145/3276528>.

[P21] LUNGU, Mircea. Designing personalized learning environments through monitoring and guiding user interactions with code and natural language. **The 1st ACM SIGSOFT International Workshop on Education through Advanced Software Engineering and**

Artificial Intelligence (EASEAI) [S.l.]: ACM Press, 2019. Disponível em: <http://dx.doi.org/10.1145/3340435.3342724>.