UNIVERSIDADE FEDERAL DO AMAZONAS - UFAM

INSTITUTO DE COMPUTAÇÃO - ICOMP

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA - PPGI

Representação de Dados Heterogêneos em Cenário de Poucos Dados Aplicada a Automação de Teste de Software Através de Redes Siamesas

Yan Rodrigo Da Silva Soares

Manaus - AM

Outubro de 2023

Yan Rodrigo Da Silva Soares

Representação de Dados Heterogêneos em Cenário de Poucos Dados Aplicada a Automação de Teste de Software Através de Redes Siamesas

Dissertação de Mestrado submetida à avaliação, como requisito parcial, para a obtenção do título de Mestre em Informática no Programa de Pós-Graduação em Informática - PPGI do Instituto de Computação - ICOMP da Universidade Federal do Amazonas - UFAM.

Orientador(a)

André Luiz da Costa Carvalho, Dr.

Universidade Federal do Amazonas - UFAM

Instituto de Computação - IComp

Manaus - AM

Outubro de 2023

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Soares, Yan Rodrigo da Silva
S676r Representação de dados heterogêneos em cenário de poucos dados aplicada a automação de teste de software através de redes siamesas / Yan Rodrigo da Silva Soares . 2023
114 f.: il. color; 31 cm.

Orientador: André Luiz da Costa Carvalho
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Redes siamesas. 2. Bert. 3. Bloom. 4. Fusão de ranking. 5. Dados sintéticos. I. Carvalho, André Luiz da Costa. II. Universidade Federal do Amazonas III. Título

Ministério da Educação
Universidade Federal do Amazonas
Coordenação do Programa de Pós-Graduação em Informática

FOLHA DE APROVAÇÃO

"REPRESENTAÇÃO DE DADOS HETEROGÊNEOS EM CENÁRIO DE POUCOS DADOS APLICADA A AUTOMAÇÃO DE TESTE DE SOFTWARE ATRAVÉS DE REDES SIAMESAS"

YAN RODRIGO DA SILVA SOARES

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Dr. André Luiz da Costa Carvalho - PRESIDENTE

Prof. Dr. Edleno Silva de Moura - MEMBRO INTERNO

Prof. Dr. Leandro Balby Marinho - MEMBRO EXTERNO

Manaus, 02 de outubro de 2023

Documento assinado eletronicamente por **Leandro Balby Marinho**, **Usuário Externo**, em 16/10/2023, às 06:11, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Edleno Silva de Moura**, **Professor do Magistério Superior**, em 18/12/2023, às 15:19, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufam.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1714714** e o código CRC **D868354E**.

Avenida General Rodrigo Octávio, 6200 - Bairro Coroado I Campus Universitário Senador Arthur Virgílio Filho, Setor Norte - Telefone: (92) 3305-1181 / Ramal 1193
CEP 69080-900, Manaus/AM, coordenadorppgi@icomp.ufam.edu.br

Referência: Processo nº 23105.043101/2023-05 SEI nº 1714714

À Antônio José da Silva Pereira (*In Memoriam*), Jairo da Silva Pereira (*In Memoriam*) e Verônica Santos Oliveira (*In Memoriam*), tios queridos que faleceram no decorrer do desenvolvimento dessa pesquisa.

Agradecimentos

À Deus, pois sem ele nada é possível.

À minha mãe que jamais nos deixou faltar nada, que sempre fez de tudo para sustentar três filhos sozinha. À ela eu dedico a minha vida e as minhas conquistas.

Ao meu orientador Professor Dr. André Carvalho que sempre me ajudou, me incentivou, me tranquilizou, me ensinou, e me deu todo apoio e dedicação para que eu conseguisse concluir essa pesquisa. Professor, você é demais, sem a sua ajuda nada disso seria possível, obrigado de coração por tudo!

À professora Eulanda Santos que acreditou em mim na minha entrada no PPGI e iniciou um projeto anterior mesmo sem ter me conhecido como aluno, eu agradeço pelo voto de confiança dado.

À minha esposa Pâmela Soares por sempre me apoiar e acreditar em mim em todas as escolhas e decisões na minha vida. Por me ajudar e entender minha ausência nesses anos e também por me dar o maior presente da minha vida, nossa filha Maria Yasmin. E é claro, não menos importante, agradeço por revisar essa dissertação com o maior zelo e cuidado.

Aos meus professores que me guiaram no decorrer da minha vida, do ensino fundamental ao mestrado, sou grato à vocês por me darem o bem mais precioso de todos: o conhecimento.

Aos meus colegas de projeto, Arthur, Maikon, Vinicius, Adamor e Hygo pelo suporte e pelas conversas que me auxiliaram ainda mais no desenvolvimento dessa pesquisa, e consequentemente no meu desenvolvimento acadêmico.

Ao ICOMP e a UFAM por incentivar a pesquisa científica, e disponibilizar os

recursos necessários para o desenvolvimento da mesma.

À minha família, pelo apoio e confiança que sempre me deram em toda a minha vida.

E aos meus amigos em geral, por todo o suporte.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Este trabalho foi parcialmente financiado pela Fundação de Amparo à Pesquisa do Estado do Amazonas – FAPEAM – por meio do projeto POSGRAD.

"O que sabemos é uma gota; o que ignoramos é um oceano."

Sir Isaac Newton

Representação de Dados Heterogêneos em Cenário de Poucos Dados Aplicada a Automação de Teste de Software Através de Redes Siamesas

Autor: Yan Rodrigo Da Silva Soares

Orientador: André Luiz da Costa Carvalho, Dr.

Resumo

O lançamento de novas versões do sistema operacional *Android* induz os fabricantes de dispositivos móveis a introduzirem suas próprias atualizações para garantir a compatibilidade e a qualidade do *software*. No entanto, para assegurar a sua qualidade, é necessário conduzir testes rigorosos no mesmo, o que frequentemente implica em despesas crescentes. Desta forma, existe uma clara necessidade de automatizar ao máximo esse processo. Para isto, as empresas podem dispor de um *framework* que inclui uma série de comandos de automação, destinados a realizar tarefas simples de teste. Nesse contexto, um operador é encarregado de ler a descrição de um caso de teste e selecionar o comando correspondente. Diante dessa problemática, o objetivo deste trabalho é ajudar os operadores na busca por comandos, onde para cada passo de um caso de teste, procura-se o comando que executa esta ação. Foram utilizados redes siamesas, combinadas com MLM (*Masked Language Model*), para representar tanto os passos quanto os comandos de automação no mesmo espaço vetorial. Isso nos permite buscar comandos com base na similaridade de cosseno. Nos propomos a usar uma função de perda que aproxime os passos de teste de seus comandos correspondentes, de modo que fiquem próximos no espaço latente de representação. Além disso, foi incorporado o uso da BLOOM, um modelo de linguagem, para gerar dados sintéticos

que auxiliam na busca por comandos quando não há um par correspondente de passo de teste. Para representar os dados, foi utilizado o modelo RoBERTa e por fim, aplicado o *LambdaMART* para realizar uma fusão de *ranking* nas sugestões de classificação dadas nos *rankings*, criando assim um *ranking* enriquecido. Os resultados finais foram muito promissores em ambos os experimentos propostos, avaliados por meio das métricas MRR, MAP e *HitRate*, onde houve uma média de 0.58 para o MRR no primeiro experimento e 0.31 no segundo experimento, concluindo que a proposta analisada é eficaz ao recomendar os comandos corretos nas posições mais altas do *ranking* recomendado.

Palavras-chave: Redes Siamesas, BERT, BLOOM, Similaridade de Cosseno, Fusão de *Ranking*, Dados Sintéticos.

Representação de Dados Heterogêneos em Cenário de Poucos Dados Aplicada a Automação de Teste de Software Através de Redes Siamesas

Autor: Yan Rodrigo Da Silva Soares

Orientador: André Luiz da Costa Carvalho, Dr.

Abstract

The release of new versions of the Android operating system leads mobile device manufacturers to introduce their own updates to ensure software compatibility and quality. However, to ensure its quality, it is necessary to conduct rigorous testing, which often implies increasing costs. Therefore, there is a clear need to automate this process as much as possible. For this, companies can have a framework that includes a series of automation commands, designed to perform simple test tasks. In this context, an operator is responsible for reading the description of a test case and selecting the corresponding command. Faced with this problem, the objective of this work is to help operators in the search for commands, where for each step of a test case, the command that executes this action is sought. Siamese networks, combined with MLM (Masked Language Model), were used to represent both the steps and the automation commands in the same vector space. This allows us to search for commands based on cosine similarity. We propose to use a loss function that approximates the test steps to their corresponding commands, so that they are close in the latent representation space. In addition, the use of BLOOM, a language model, was incorporated to generate synthetic data that helps in the search for commands when there is no corresponding test step pair. To represent the data, the RoBERTa model was used and finally, LambdaMART

was applied to perform a ranking fusion in the classification suggestions given in the rankings, thus creating an enriched ranking. The final results were very promising in both the proposed experiments, evaluated using the MRR, MAP, and HitRate metrics, where there was an average of 0.58 for MRR in the first experiment and 0.31 in the second experiment, concluding that the proposal analyzed is effective in recommending the correct commands in the highest positions of the recommended ranking.

Keywords: *Siamese Networks, BERT, BLOOM, Cosine Similarity, Ranking Fusion, Synthetic Data*.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AM Aprendizado de Máquina

API *Application Programming Interface*

BERT *Bidirectional Encoder Representations for Transformers*

BLOOM *BigScience Large Open-science Open-access Multilingual Language Model*

BPE *Byte-Pair Encoding*

IA Inteligência Artificial

LLM *Large Language Model*

LSTM *Long Short-Term Memory*

LTR *Learning to Rank*

MaLSTM *Manhattan Long Short-Term Memory*

MAP *Mean Average Precision*

ML *Machine Learning*

MLM *Masked Language Model*

MRR *Mean Reciprocal Rank*

PLN Processamento de Linguagem Natural

QA *Quality Assurance*

RI Recuperação de Informação

SUMÁRIO

1

INTRODUÇÃO

1.1 Motivação e Contexto

Em tarefas de Aprendizado de Máquina (AM) e Inteligência Artificial (IA), os dados são centrais, especialmente quando devidamente rotulados. Contudo, a coleta e análise de dados em um volume grande o suficiente para essas tarefas não é trivial. Em tarefas mais específicas da indústria, o sigilo comercial e as características de isolamento naturais das empresas podem dificultar a extração de grandes conjuntos de dados, especialmente dados relacionados a uso de ferramentas, já que muitas das tarefas podem ser baseadas em sistemas internos de acesso exclusivo da empresa.

Este trabalho apresenta os desafios e técnicas que foram implantadas em uma aplicação de automação de execuções de casos de teste de *software Android*.

Todos os anos a empresa *Google*, uma empresa multinacional de serviços online e software, lança novas versões do sistema operacional *Android* levando os fabricantes de *smartphones Android* a também lançarem suas versões atualizadas do sistema para seus telefones atualmente no mercado, com ajustes para se adequar a novas funcionalidades do *Android* com aplicativos proprietários da empresa e modificações no sistema *Android* base.

Para garantir a qualidade de cada lançamento, testes exaustivos são realizados em grande escala em todas as novas versões para que os produtos sejam entregues com versões estáveis na medida do possível. Nesse contexto, os Testes de *Software* são cruciais para garantir a qualidade e a boa execução de um *software* em um ambiente de produção. Eles normalmente são executados por meio de **casos de teste**, conjuntos

de passos a serem seguidos durante o teste, geralmente descritos em uma linguagem natural como o inglês, para avaliar uma determinada funcionalidade.

Estes casos de teste normalmente são executados por operadores humanos, portanto a sua automação permitiria a sua execução de maneira rápida e eficiente. Em casos específicos, como dispositivos proprietários ou situações menos corriqueiras, é comum que no processo de automação a utilização de um *framework* de automação, que fornece ferramentas e funcionalidades que facilitam a escrita, a execução e a manutenção dos testes automatizados.

Para que isso seja possível, é comum que estes *frameworks* disponibilizem uma série de **comandos de automação**, que são implementações de passos específicos de um caso de teste desenvolvidos em alguma linguagem de programação. Os passos específicos que geralmente são instruções em linguagem natural são denominados **passos de teste**.

Um *framework* de automação pode conter milhares de comandos diferentes disponíveis, e os programadores de automação de casos de teste são responsáveis por ler a descrição em linguagem natural de um determinado caso de teste e implementá-lo usando os comandos corretos entre a grande massa de comandos possíveis.

Conforme mostrado na Figura 1, para a automatização de um caso de teste geralmente um automatizador lê um Caso de Teste (escrito em linguagem natural) e a partir dos seus passos de teste, busca no *framework* de automação os comandos que executam cada passo de teste. A partir disso, tendo os comandos corretos, é criado um *script* em uma linguagem de programação (no nosso caso em Python) que automatiza todo o caso de teste. Após isso, um testador que necessitar executar um caso de teste pode rodar o script que foi desenvolvido para a realização do mesmo.

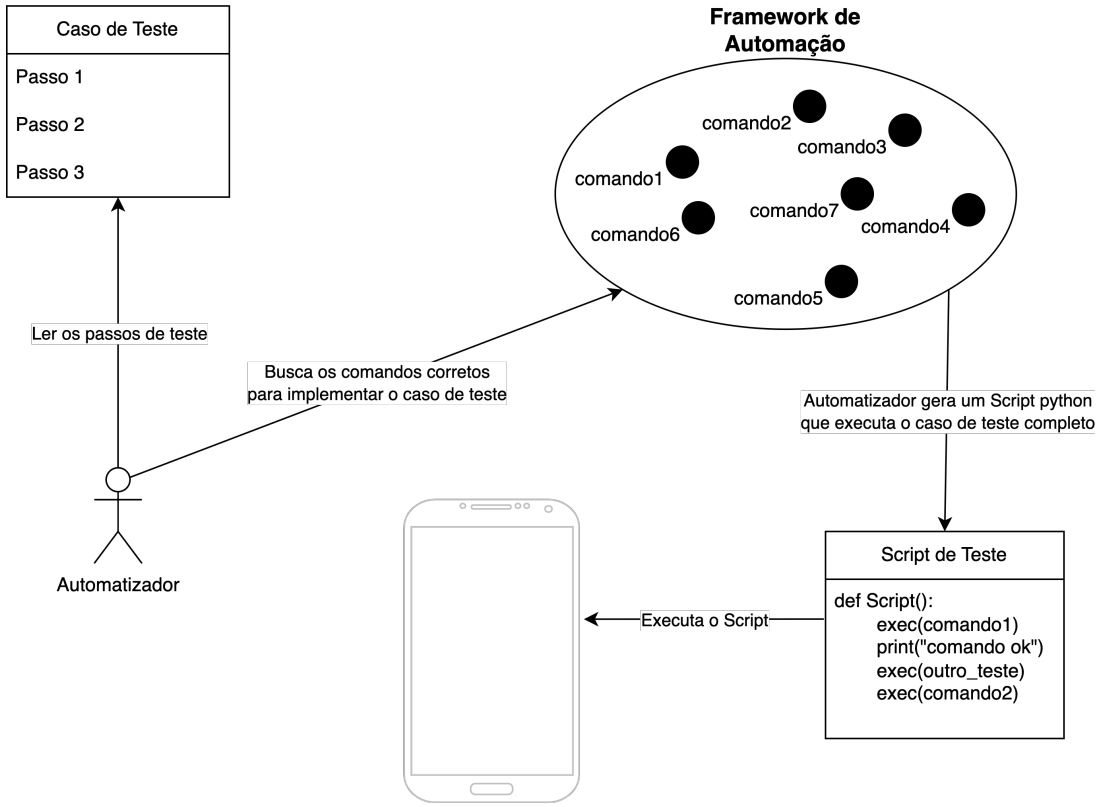

Figura 1 – Diagrama do processo de teste

Fonte: Próprio Autor

Esta abordagem, apesar de bem sucedida, é custosa. Conforme o escopo de um *software* a ser testado cresce, o número de testes a serem feitos aumenta, e muitas vezes de forma não linear, já que cada componente novo adicionado deve ser testado não apenas em si, mas também nas suas interações com outros componentes previamente implementados. Desta forma, o volume de testes a serem feitos a cada atualização acaba levando a um aumento expressivo nos custos para manutenção e atualização de *softwares*.

Além disso, a equipe de implementação também tem que ter um profundo conhecimento do *software* a ser testado, e ainda sofrem com o fato da descrição dos passos de teste ser feita em linguagem natural, já que diferentes pessoas podem descrever o mesmo passo de formas radicalmente diferentes, o que é inerente em qualquer tarefa que receba como entrada texto em linguagem natural. Este problema é potencializado quando há times de teste oriundos de diferentes países, muitos sem ter inglês como a primeira língua, escrevendo os passos.

A elaboração e automação de testes para este tipo de funcionalidade adiciona uma dimensão extra de complexidade ao problema devido à características em geral não determinísticas do treinamento de algoritmos de aprendizagem de máquina.

Não obstante os problemas acima descritos, ainda existe o problema do retrabalho. Dada a natureza *ad hoc* do processo de geração de casos de teste, muitas vezes os programadores têm que refazer implementações para passos que eles já implementaram anteriormente. Tudo isto significa que existe um retrabalho oriundo unicamente da não sistematização da geração destes casos de teste.

Em um mundo ideal, uma vez que um passo fosse implementado, esta mesma implementação poderia ser utilizada em futuros casos de teste que tivessem passos similares, reaproveitando trabalhos anteriores e diminuindo o retrabalho.

Isso torna a automação de casos de teste uma empreitada complexa e cara: devido a ser uma ferramenta proprietária, não é trivial encontrar (ou treinar) programadores para aprender os detalhes do seu uso. Além disso, o tamanho e a diversidade do conjunto de casos de teste também apresentam suas dificuldades, uma vez que o conhecimento obtido a partir da automação de casos de teste relacionados à câmera, por exemplo, não se traduz para domínios de testes diferentes, como o navegador de *internet*. Estes custos relacionados à formação de programadores são outra dificuldade no processo de automação.

Uma maneira de resolver estes problemas é usar técnicas de recuperação de informação para buscar, com base em um passo de teste, um comando de automação específico que o implementa. No entanto, isso não é trivial: enquanto os passos são escritos em linguagem natural, os comandos são escritos como operações de código fonte, que são evidentemente de um domínio diferente.

Essa incompatibilidade entre domínios torna difícil encontrar os comandos corretos para automatizar cada passo, uma vez que um passo pode ser escrito em diferentes formas e escolhas de palavras vastamente diferentes, o que não é apenas possível, mas esperado por empresas mundiais, onde o *Quality Assurance* (QA) é realizado por equipes de diferentes países e muitas vezes não têm o inglês como primeira língua.

Uma solução seria os humanos criando manualmente descrições textuais e possí-

veis passos para cada comando. No entanto, é necessário conhecimento especializado do *framework* de automação para essa tarefa, e o número de comandos (neste caso, em cerca de 30 mil) aumentaria os custos dessa solução.

E mesmo com descrições geradas por humanos, como os passos nesse caso são normalmente descrições textuais muito curtas e diretas, encontrar o comando correto nesse cenário é ainda mais difícil devido à falta de informações para simplificar a correspondência entre um passo de teste e o comando que o atualiza.

Além disso, como mencionado anteriormente, entre os principais desafios deste trabalho, é a busca dos comandos de automação corretos em um ambiente com dados muito limitados disponíveis e com domínios distintos.

Como solução, propõe-se implantar redes neurais siamesas para representar tanto os passos de teste quanto os comandos em um único espaço de representação compartilhado, juntamente com o uso de *Large Language Model* (LLM) para gerar descrições sintéticas de passos de teste de automação para estes conjuntos de dados de comandos, usando técnicas de aprendizado de poucas amostras, a fim de aumentar e enriquecer as informações sobre um comando, para melhorar as classificações dos métodos de recuperação.

Assim, este trabalho propõe também o uso de métodos de enriquecimento de dados utilizando modelos LLM para realizar o aumento de dados, a fim de criar um mecanismo de busca que, dado um passo de teste, recupere uma lista de comandos candidatos para automatizá-lo e classificá-los por relevância computada. A Figura 2 mostra o diagrama geral das etapas citadas.

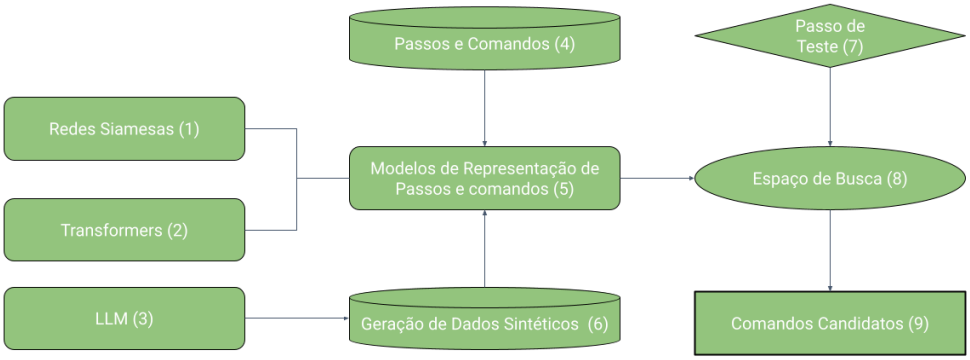

Figura 2 – Diagrama das etapas a serem seguidas no trabalho

Fonte: Próprio Autor

A Figura 2 acima mostra os passos gerais a serem realizados no trabalho: Para representar os passos e os comandos (5) serão utilizados modelos com base na arquitetura Transformer (2) (VASWANI et al., 2017) treinando-os em uma arquitetura de Redes Siamesas (1), com o objetivo de aproximar os passos e os comandos correspondentes. Para comandos que não tem passo associado, uma LLM (3) será usada para criação de dados sintéticos (6), e esses dados sintpeticos juntamente com a base de passos e comandos (4) será utilizada no modelo de representação para gerar o espaço de busca (8). Por fim, quando um passo de teste (7) for dado como entrada será realizado um busca via distância de cosseno no espaço vetorial de busca(8) gerando assim a lista de comandos candidatos (9).

1.2 Hipótese

A hipótese desse trabalho é definida como: ”Representar passos de teste e comandos de automação no mesmo espaço vetorial usando redes siamesas e aproximando-os utilizando uma função de perda pode melhorar a busca de comandos, dado um passo de teste como entrada”

1.3 Objetivos

1.3.1 Objetivo Geral

Recuperar comandos de automação candidatos para a automação de um caso de teste dada a descrição textual dos seus passos de teste.

1.3.2 Objetivos Específicos

Para alcançar o Objetivo Geral, os objetivos específicos definem-se como:

- Representar passos de teste e comandos no mesmo espaço vetorial de busca;
- Criar representações sintéticas de passos de teste para comandos que não tem passos de teste anteriormente associados a ele;
- Utilizar técnicas de fusão de *ranking* para unir os *rankings* gerados pelo nosso modelo a fim de obter a melhor classificação.

1.4 Organização do Trabalho

Esta dissertação está organizada da seguinte forma: O capítulo 2, apresenta o referencial teórico para entendimento das técnicas utilizadas neste trabalho.

O capítulo 3, apresenta os trabalhos relacionados, expondo vantagens e desvantagens das técnicas já existentes.

O capítulo 4, descreve o modelo proposto e o capítulo 5, apresenta os resultados finais da pesquisa.

Para finalizar, o capítulo 6 apresenta as considerações finais e trabalhos futuros que poderão ser seguidos futuramente.

2

Referencial Teórico

2.1 Processamento de Linguagem Natural

Processamento de Linguagem Natural (PLN) é uma área da Ciência da Computação que estuda o desenvolvimento de programas de computador que analisam, reconhecem e/ou geram textos em linguagens humanas (ou linguagens naturais). O PLN não é uma tarefa trivial devido à rica ambiguidade da linguagem natural. Essa ambiguidade o torna diferente do processamento das linguagens de programação de computador, as quais são formalmente definidas evitando, justamente, a ambiguidade (ZILIO, 2010).

Em PLN, devido à rica variedade de idiomas no mundo e às suas ambiguidades inerentes, tem-se diversos objetivos a serem estudados, tais como: Tradução de Idiomas, Recuperação de Informação (a partir de textos), entre outros. A Tradução de Idiomas é uma das tarefas mais utilizadas, devido à clara necessidade de comunicação entre pessoas de diferentes países. Em Recuperação de Informação (RI), a grande capacidade que os modelos de PLN têm de extrair informações semânticas e sintáticas do texto ajuda a melhorar os mecanismos de busca.

Nas subseções abaixo, serão mostradas algumas técnicas comumente utilizadas.

2.1.1 Pré-Processamento de Sentenças

Antes dos dados serem utilizados para a entrada dos modelos de redes neurais, é necessário tratá-los para que os modelos possam realizar as operações de forma correta e também que não haja ruído na entrada. Algumas das técnicas mais utilizadas são a *Tokenização*, *Padding*, *Embeddings* de Palavras, Remoção de *Stopwords*, Inclusão de *tokens* especiais (início e fim de frases, palavras desconhecidas, *etc.*), *Stemming*, entre outras. Algumas dessas técnicas serão explicadas a seguir.

2.1.2 Tokenização

A *Tokenização* consiste em dividir o texto em palavras, e a partir daí gerar os *tokens* correspondentes a cada uma dessas palavras e o texto é dividido utilizando um delimitador pré-estabelecido. (DEVLIN et al., 2018).

A *Tokenização* é uma das etapas mais importantes do pré-processamento de texto porque é a partir dela, que pode-se eliminar ruídos nos dados fazendo análises estatísticas a respeito do vocabulário gerado. Geralmente, a *tokenização* é realizada dividindo o texto pelos espaços em branco, mas atualmente várias pesquisas estão sendo realizadas para melhorar a forma de *tokenizar* visando ganho nos modelos de redes neurais (DEVLIN et al., 2018).

Pode-se informar o *Byte Pair Encoding* (BPE) (GAGE, 1994) como uma das técnicas mais utilizadas, com o advento do modelo *Bidirectional Encoder Representations for Transformers* (BERT).

2.1.3 *Padding*

Um problema muito comum quando trabalha-se com texto são os diferentes tamanhos de sentenças utilizados como treinamento do modelo. No caso das redes neurais, é preciso garantir que todas as entradas sejam do mesmo tamanho.

Para isso é utilizado o *padding*, que acrescenta *tokens* "nulos" para que as sentenças estejam todas do mesmo tamanho para a entrada do modelo de rede neural, ou seja,

se na base de dados têm-se várias frases, e a maior frase tem tamanho X, todas as frases ficarão do tamanho X (sendo adicionado *tokens* nulos no fim da frase até que se tenha o tamanho X). (RISHITA; RAJU; HARRIS, 2019)

2.1.4 Vetorização

A vetorização é uma técnica para transformar um texto em representações numéricas para que o computador possa entendê-las. Geralmente consiste em transformar o texto em um vetor numérico, em que os valores podem ser calculados pela frequência de palavras usando por exemplo o TF-IDF (BAFNA; PRAMOD; VAIDYA, 2016), ou via extração de características semânticas com *Word Embeddings* (ALMEIDA; XEXÉO, 2019)) ou mesmo por codificação simples utilizando *bag-of-words*. Neste trabalho, foi utilizado a vetorização via *Embeddings*.

2.1.5 *Embeddings*

Embeddings são espaços vetoriais, sendo cada vetor uma palavra de um determinado idioma. A representação dessa forma reproduz um papel vital nas tarefas de PLN, pois captura informações semânticas e sintáticas das palavras podendo assim ser medida, por exemplo, a semelhança entre as palavras (LIU et al., 2015).

As vantagens estão na representação do vetor, já que os vetores são menores em comparação a *one-hot encoding* e a representações das palavras apresentam significado semântico, podendo ser aprendidas no treino da rede neural.

2.2 *Large Language Models*

Large Language Models (LLM) são modelos de Inteligência Artificial (IA) capazes de processar e entender a linguagem natural. Esses modelos são projetados para imitar a maneira como os seres humanos aprendem e entendem a linguagem, usando grandes quantidades de dados para treinar redes neurais profundas.

Esses modelos são capazes de gerar texto coerente e natural, responder a perguntas com precisão e realizar outras tarefas relacionadas à linguagem natural. Os LLM's mais avançados atualmente disponíveis utilizam arquiteturas de rede neural profundas, como a *Transformer*, que permite ao modelo capturar relações semânticas complexas entre palavras e frases.

Um dos LLM's mais conhecidos é a *Generative Pre-Trained Transformer* - GPT desenvolvida pela *OpenAI* (RADFORD et al., 2018). Esses modelos foram treinados em conjuntos de dados maciços de texto, como livros, artigos e *sites*, para aprender os padrões e estruturas da linguagem. A última versão, GPT-4, é um dos LLM's mais poderosos disponíveis atualmente (OPENAI, 2023).

Esses modelos têm tido um impacto significativo em várias áreas, incluindo processamento de linguagem natural, tradução automática, geração de texto, análise de sentimento, entre outras.

Os LLM's têm uma ampla gama de aplicações, desde processamento de linguagem natural e reconhecimento de fala até *chatbots* e assistentes virtuais. Eles podem ser usados para analisar e resumir grandes quantidades de texto, gerar novo conteúdo e até mesmo traduzir entre idiomas.

No entanto há preocupações com as implicações éticas de criar máquinas capazes de gerar linguagem semelhante à humana, bem como o potencial de uso indevido. Os LLM's também apresentam desafios significativos, incluindo o viés nos dados de treinamento e a capacidade de disseminar informações falsas ou enganosas. Além disso, o treinamento desses modelos requer grandes quantidades de recursos computacionais e energia, o que levanta preocupações sobre o impacto ambiental. Como esses modelos são capazes de gerar texto coerente e natural, é possível usá-los para criar conteúdo falso ou enganoso que pareça autêntico. À medida que a tecnologia dos LLM's avança, é importante considerar cuidadosamente os benefícios e desafios associados a essa tecnologia e desenvolver estratégias para maximizar seus benefícios e minimizar seus riscos.

2.2.1 LLM's em Recuperação de Informação

Falando um pouco sobre a aplicação dos LLM's em Recuperação de Informação (RI), esses modelos têm sido amplamente utilizados em tarefas de recuperação de informação, especialmente em sistemas de busca de texto, e aumento de dados (BONIFACIO et al., 2022). Os LLM's são capazes de capturar informações semânticas e contextuais em documentos e consultas, o que permite gerar representações mais precisas e relevantes dos mesmos.

A utilização de LLM's na recuperação de informação tem apresentado resultados promissores em diversos estudos. Por exemplo, o *Google* incorporou recentemente o modelo BERT em seu algoritmo de busca, o que melhorou significativamente a qualidade dos resultados apresentados aos usuários (MARTIN, 2023). Além disso, os LLM's podem ser utilizados para gerar consultas mais precisas e relevantes, bem como para melhorar a reescrita de consultas.

Outra vantagem dos LLM's na recuperação de informação é a possibilidade de transferência de conhecimento. Modelos pré-treinados em grandes *corpus* de texto podem ser adaptados para tarefas específicas de recuperação de informação, o que reduz a necessidade de grandes conjuntos de dados anotados (BONIFACIO et al., 2022).

Além disso, os LLM's podem ser combinados com outras técnicas de recuperação de informação, como fusão de *ranking*, para gerar resultados ainda mais precisos e relevantes. A combinação de LLM's com outras técnicas também pode ajudar a contornar limitações dos modelos, como a falta de cobertura em determinados domínios.

2.2.2 BLOOM

O *BigScience Large Open-science Open-access Multilingual Language Model* (BLOOM) é um modelo de linguagem de grande escala desenvolvido pelo projeto *BigScience*, que é uma iniciativa de pesquisa em inteligência artificial com o objetivo de construir modelos de linguagem de grande escala e com alta qualidade em diversas línguas (SCAO et al., 2022).

O BLOOM é baseado no *Transformer*. O modelo é treinado em uma grande

quantidade de dados multilíngues disponíveis na internet e em fontes abertas. Isso permite que o BLOOM tenha um bom desempenho em várias tarefas de processamento de linguagem natural, como tradução automática, análise de sentimento, geração de texto, entre outras (SCAO et al., 2022).

Uma das principais características do BLOOM é sua capacidade de suportar várias línguas. Isso é possível graças à grande quantidade de dados multilíngues utilizados durante o treinamento do modelo. Essa capacidade é importante porque muitas vezes é necessário trabalhar com textos em várias línguas em diferentes contextos, e o BLOOM pode ajudar a realizar essas tarefas de forma mais eficiente.

O BLOOM também é um modelo de linguagem aberto e de acesso livre, o que significa que qualquer pessoa pode acessá-lo e usá-lo para suas próprias necessidades de processamento de linguagem natural. Isso é importante porque muitas vezes os modelos de linguagem disponíveis no mercado são proprietários e podem ser caros para empresas e indivíduos que desejam usá-los.

Como já citado, uma das principais vantagens do BLOOM é a sua capacidade multilíngue, o que significa que pode entender e gerar texto em vários idiomas diferentes. Isso é particularmente importante em um mundo cada vez mais globalizado, onde a comunicação transcultural é cada vez mais comum.

O BLOOM tem um desempenho muito forte em várias tarefas de linguagem natural, como tradução automática, geração de texto e classificação de texto, o que o torna uma ferramenta valiosa para pesquisadores em várias áreas. Além disso, a equipe da *BigScience* está comprometida em manter o modelo atualizado e melhorá-lo continuamente, o que significa que ele deve continuar sendo uma ferramenta valiosa para os pesquisadores no futuro.

No entanto, é importante notar que o modelo BLOOM não é perfeito e ainda há desafios a serem superados. Um dos principais desafios é a necessidade de recursos computacionais significativos para executar o modelo, o que pode limitar sua acessibilidade para pesquisadores e empresas com recursos limitados.

Além disso, o modelo ainda não é perfeito em todas as tarefas de linguagem natural e há desafios a serem superados na compreensão de contextos complexos e

nuances linguísticas, além da questão ética.

2.3 Redes Siamesas

Redes Siamesas são uma arquitetura de redes neurais que são utilizadas principalmente em tarefas de comparação e verificação, tais como verificação facial, comparação de assinaturas e detecção de anomalias. Elas são chamadas ”siamesas” porque têm duas ou mais entradas idênticas que compartilham os mesmos pesos.

O objetivo geral de uma rede siamesa é aprender como diferenciar entre duas entradas. Por exemplo, se for dada a tarefa de verificar se duas imagens são da mesma pessoa, a rede siamesa será treinada para entender e aprender as diferenças e semelhanças entre as imagens da mesma pessoa e de pessoas diferentes.

Uma rede siamesa simples tem duas entradas, para cada uma das duas entradas que você quer comparar. Por exemplo, essas entradas poderiam ser duas imagens de rostos. Cada entrada passa por uma série de camadas (que são idênticas e compartilham os mesmos pesos) que extraem características da entrada, em um processo chamado de codificação.

A distância entre os vetores de características extraídos são então computadas através de algum tipo de função de distância. Geralmente, essa medida de distância é a distância euclidiana ou a distância de cosseno, mas pode ser qualquer medida de distância adequada ao problema em questão. O resultado final é essa medida de distância, que é usada para determinar se as duas entradas são semelhantes ou não.

Um exemplo deste tipo de arquitetura é ilustrado na Figura 3. Nesta arquitetura, duas redes (Rede A e Rede B) trabalham lado a lado para representar um par de objetos, com o objetivo de aproximar vetorialmente as duas entradas (Entrada A e Entrada B), diminuindo a distância vetorial entre entradas similares e aumentando a distância para aquelas distintas, por meio de uma função de distância usada como função de perda para treinamento. Além disso, existem versões desta arquitetura em que é possível comparar mais de duas entradas ao mesmo tempo.

A ideia central da rede neural siamesa é utilizar redes neurais conjuntas que

compartilhem pesos w, e representações de elementos distintos em um mesmo espaço de latente de características, (ROCHA; CARVALHO, 2021).

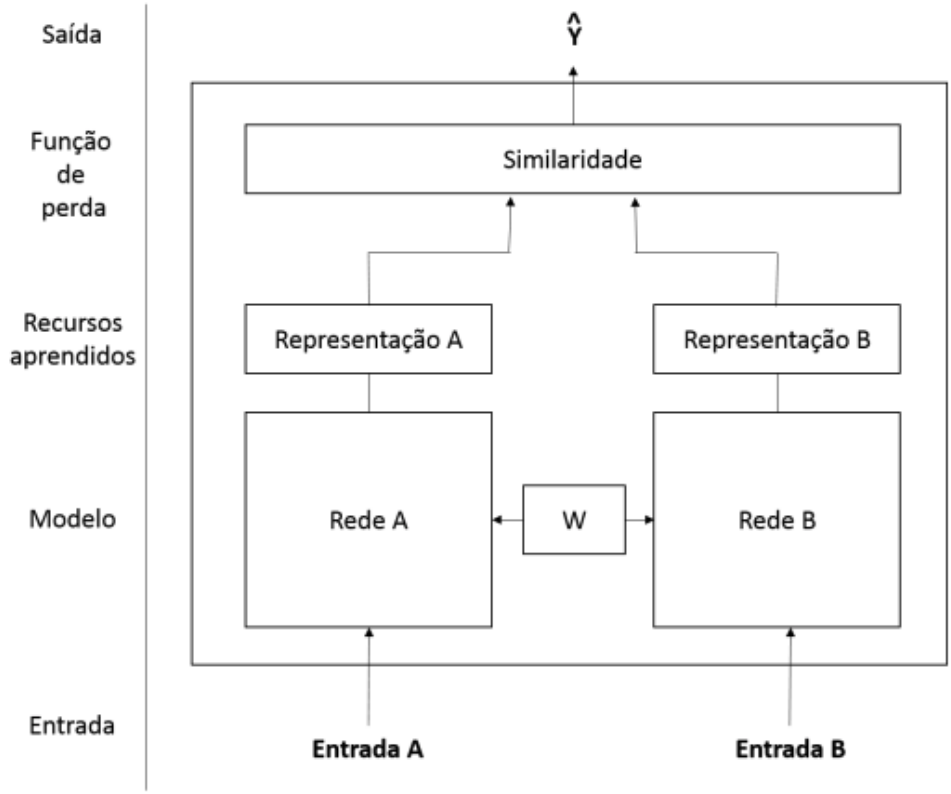

Figura 3 – Arquitetura de Redes Siamesas para comparação

Fonte: Traduzido de (ROCHA; CARVALHO, 2021)

A função de perda é uma parte crucial do treinamento de qualquer rede neural, e nas redes siamesas, uma função de perda comum é a "*Contrastive Loss*". Ela tenta garantir que a distância entre exemplos de classes semelhantes seja pequena e que a distância entre exemplos de classes diferentes seja grande.

Outra função de perda comum é a "*Triplet Loss*". Em vez de apenas pares de exemplos, a *Triplet Loss* considera trios de exemplos: um exemplo âncora, um exemplo positivo (da mesma classe que o âncora) e um exemplo negativo (de uma classe diferente). A ideia é garantir que o exemplo positivo esteja mais próximo do âncora do que o exemplo negativo por uma certa margem.

Uma das grandes vantagens das redes siamesas é a sua capacidade de lidar com pequenas quantidades de dados. Em muitos casos, têm-se um grande número de exemplos para cada classe. As redes siamesas são eficazes nesses casos porque estão

focadas na aprendizagem das diferenças entre pares ou trios de exemplos, em vez de tentar aprender a classe de cada exemplo individualmente.

Além disso, as redes siamesas podem ser eficientes em termos de computação, pois a codificação para várias entradas pode ser compartilhada.

2.4 Recuperação de Informação

Recuperação de Informação (RI) é uma área da Ciência da Computação que concentra-se em prover aos usuários um acesso mais fácil a informações do seu interesse. Em suma, essa área trata da representação, armazenamento, organização e acesso aos itens de informação tais como documentos, páginas *web*, *etc*. A área cresceu muito, e atualmente inclui modelagem, classificação textual, visualização de dados entre outros. (BAEZA-YATES; RIBEIRO-NETO, 2013).

Nas próximas subseções explica-se um pouco sobre o foco deste trabalho, que é a busca semântica. Também fala-se um pouco sobre o método de busca e a fusão de *ranking* e por fim fala-se sobre as métricas utilizadas neste trabalho.

2.4.1 Busca Semântica

A busca semântica é uma forma de busca que tenta entender o significado das palavras e frases em uma consulta de pesquisa, em vez de simplesmente procurar por padrões de texto. Isso permite que os motores de busca retornem resultados mais relevantes para a consulta do usuário. A busca semântica visa melhorar a qualidade dos resultados das buscas tentando entender a intenção do usuário, assim como o significado contextual dos termos de busca (ALVIM, 2017).

Entre outras técnicas a análise semântica se destaca. Análise Semântica é o processo de entender o significado das palavras e frases em uma sentença. Os motores de busca podem usar a análise semântica para identificar o significado das consultas de pesquisa e retornar resultados mais relevantes.

Na busca semântica, as consultas são interpretadas em um contexto mais amplo,

levando em consideração o significado das palavras e sua relação com outras palavras e conceitos. Isso permite que os resultados da pesquisa sejam mais precisos e relevantes para o usuário. (KLUSCH et al., 2016)

2.4.2 Distância de Cosseno

A similaridade por cosseno é uma medida da similaridade de entre dois vetores num espaço vetorial que avalia o valor do cosseno do ângulo compreendido entre eles (BAEZA-YATES; RIBEIRO-NETO, 2013). Sua fórmula pode ser descrita na Figura 4 abaixo:

$$Sim_{cos}(\overrightarrow{x}, \overrightarrow{y}) = \frac{\overrightarrow{x}.\overrightarrow{y}}{||\overrightarrow{x}||||\overrightarrow{y}||}$$

Figura 4 – Fórmula distância de cosseno

Fonte: Adaptada de (GEORGE; KUMAR; PANDA, 2015)

A similaridade do cosseno fornece um valor no intervalo [-1,1], porém é comum ser usada em contextos onde todos os valores são positivos, fornecendo então um valor entre [0,1].

Essa medida é usada em várias áreas onde a magnitude dos vetores não é (tão) importante quanto sua direção, em especial em busca e recuperação de informação, para medir a semelhança entre uma consulta e um documento, e em mineração de textos, onde estabelece uma métrica de semelhança entre textos, sendo aplicável tanto em algoritmos de classificação como de agrupamento.

Quando usada como medida de avaliação da similaridade de textos, cada valor do vetor indica um peso para uma palavra ou conceito usado no texto, criando um espaço vetorial, neste caso, como todos os vetores são positivos, a medida está no intervalo [0,1] (VIPIN et al., 2006; SINGHAL et al., 2001).

2.4.3 *Learning To Rank*

Learning to Rank (LTR) é uma classe de técnicas que aplicam aprendizado de máquina (AM) / *Machine Learning* (ML) supervisionado para resolver problemas de classificação.

A principal diferença entre LTR e ML supervisionado tradicional é que ML tradicional resolve um problema de previsão (classificação ou regressão) em uma única instância por vez. Por exemplo, se você estiver fazendo detecção de *spam* em *e-mail*, você examinará todos os recursos associados a esse *e-mail* e o classificará como *spam* ou não. O objetivo do ML tradicional é criar uma classe (*spam* ou não-*spam*) ou uma única pontuação numérica para essa instância. Já o LTR resolve um problema de classificação em uma lista de itens.

O objetivo do LTR é criar uma ordem ideal desses itens. Como tal, o LTR não se preocupa muito com a pontuação exata que cada item obtém, mas se preocupa mais com a ordem relativa entre todos os itens. (LIU et al., 2009)

2.4.4 Fusão de Ranking

A fusão de *ranking* é uma técnica amplamente utilizada em Sistemas de Recuperação de Informação (RI) e Sistemas de Recomendação (SR) para combinar várias listas de rankings em um único *ranking* consolidado.

O objetivo é melhorar a qualidade e a relevância dos resultados apresentados aos usuários. A fusão de *ranking* é especialmente útil quando os *rankings* individuais são gerados por diferentes algoritmos, fontes de dados ou critérios de classificação, e você deseja combinar esses *rankings* para obter uma visão mais abrangente e precisa, além de ser uma técnica poderosa que pode ser usada para melhorar a precisão e a relevância dos resultados de pesquisa (RENDA; STRACCIA, 2003).

Neste trabalho, foi utilizado para Fusão de *Ranking*, o *LambdaMART*, umas das técnicas mais usadas em LTR.

2.5 Métricas

2.5.1 *Mean Reciprocal Rank* (MRR)

Mean Reciprocal Rank (MRR) é uma métrica comumente usada para avaliar Sistemas de Recuperação de Informação (RI), levando em consideração a ordem em que os documentos relevantes são recuperados, porque quanto maior a classificação de um documento relevante, maior é o valor MRR.

MRR é a média da soma dos inversos das classificações dos documentos relevantes recuperados, portanto, pode ser considerada uma métrica para avaliar o quão bem os itens de classificação superior são precisos (CRASWELL, 2009). A fórmula do MRR é descrita na Figura 5 abaixo por:

$$MRR(Q) = \frac{1}{N_q} \sum_{i=1}^{N_q} \frac{1}{S_{correto}(R_i)}$$

Figura 5 – Fórmula *Mean Reciprocal Rank* - MRR

Fonte: (BAEZA-YATES; RIBEIRO-NETO, 2013)

Para um conjunto Q, composto por Nq consultas o MRR é a média de todos os reciprocal ranks. $S_{correto}$ é a posição do primeiro item relevante na lista de recomendações da consulta **N**.

Primeiro criou-se uma lista de recomendações para cada consulta, essas recomendações devem ser ordenadas de acordo com relevância prevista de cada item (com os itens mais relevantes primeiro). Essa lista pode ter previsões de relevância diferentes para cada item ou ser simplesmente um indicador binário.

Depois, para cada usuário, percorreu-se a lista e calculou-se o inverso multiplicativo da posição do primeiro item relevante (o "*Reciprocal Rank*"). Por fim, calculou-se a média desses valores levando em conta todas as consultas.

Caso não haja itens relevantes numa lista de resultados, foi substituído o valor do *Reciprocal Rank* por zero. O *Mean Reciprocal Rank* vai de 0 a 1, quanto mais próximo de 1, melhor.

Um valor igual a 1 significa que todos os itens na primeira posição de cada lista são relevantes. Um valor igual a zero significa que nenhum item relevante está nas listas

de resultados. Na maioria dos casos o MRR ficará entre zero e um.

2.5.2 *Mean Average Precision* (MAP)

Mean Average Precision (MAP) é uma métrica de avaliação de *ranking* que mede a precisão média dos resultados de um sistema de *ranking*. Ela é calculada como a média da precisão média de cada lista de resultados, onde a precisão de uma lista de resultados é a proporção de resultados relevantes que estão na lista.

MAP é uma métrica útil para avaliar sistemas de *ranking* que retornam listas de resultados, como sistemas de busca, sistemas de recomendação e sistemas de classificação.

O MAP pode ser definido também como:

> "Seja R_i o conjunto de documentos relevantes para a consulta q_i, $|R_i|$ o número de documentos relevantes para a consulta q_i e $R_i[k]$ uma referência ao k-ésimo documento em R_i. Então, $P(R_i[k])$ é a precisão quando o documento $R_i[k]$ é observado no ranking de q_i. Se aquele documento não for recuperado, o que é uma ocorrência comum em buscas reais, $P(R_i[k])$ assume o valor zero (que é na verdade indefinido, mas podemos supor que seja pequeno e aproximá-lo como zero) (BAEZA-YATES; RIBEIRO-NETO, 2013)."

A precisão média, MAP, de uma consulta q_i é descrita na Figura 6 abaixo:

$$MAP_i = \frac{1}{|R_i|} \sum_{k=1}^{|R_i|} P(R_i[k])$$

Figura 6 – Fórmula MAP para uma consulta q_i

Fonte: (BAEZA-YATES; RIBEIRO-NETO, 2013)

A pontuação MAP, é a média das precisões médias para um conjunto de consultas q, a fórmula geral é descrita abaixo na Figura 7:

$$MAP = \frac{1}{N_q} \sum_{i=1}^{N_q} MAP_i$$

Figura 7 – Fórmula MAP geral

Fonte: (BAEZA-YATES; RIBEIRO-NETO, 2013)

2.5.3 HitRate

HitRate é uma métrica simples para avaliar se pelo menos um documento relevante está em um *ranking* T de tamanho k gerado. A Figura 8 abaixo mostra a fórmula do *HitRate* para um *ranking* T de tamanho k

$$Hit@K = \max_{i=1..K} \begin{cases} 1, & r_i \in \mathcal{T} \\ 0, & \text{senão} \end{cases}$$

Figura 8 – Fórmula HitRate@k

Fonte: Adaptado de (RIPLEY et al., 2001)

A pontuação *HitRate* de uma consulta vai ser 1 se pelo menos um documento relevante apareceu no ranking T de tamanho k, senão a pontuação é 0. A pontuação final é a média das pontuações *HitRate* de um conjunto Q de consultas (LI et al., 2020).

3

TRABALHOS RELACIONADOS

Neste capítulo são apresentados alguns trabalhos relacionados ao objeto desta pesquisa. Para auxiliar no entendimento, os trabalhos relacionados estão subdivididos de acordo com seu ponto de intersecção com os assuntos abordados neste trabalho, a saber: Automação de Casos de Teste, Busca Semântica, Aumento de Dados e Redes Siamesas.

A ideia foi encontrar abordagens que tivessem como objetivo a similaridade textual entre dados de domínios diferentes e/ou busca semântica de dados textuais, similar a que estamos propondo neste trabalho. Lendo o título e o resumo dos artigos coletados, verificamos quais seriam os mais interessantes de analisarmos, e os artigos selecionados são mostrados no decorrer desta seção.

3.1 Redes Siamesas

Bromley et al. (1993) propôs o conceito de redes siamesas em meados de 1993. No seu trabalho o objetivo era verificar a autenticidade de assinaturas escritas utilizando duas redes idênticas para extração de características (conceito de redes siamesas) com o objetivo de medir a distancia entre duas entradas e calcular o quão similar elas são entre si.

Poksappaiboon et al. (2021) propõe um modelo para comparar duas perguntas em tailandês, analisando perguntas frequentemente feitas e perguntas de entrada de clientes via *Facebook Messenger*, usando uma rede neural Siamesa com *Manhattan Long Short-Term Memory* (MaLSTM).

O modelo visa encontrar similaridade semântica entre as perguntas. Embora o desempenho geral do modelo não seja perfeito, ele mostra resultados promissores na detecção de similaridade entre duas perguntas.

Os resultados sugerem que, com o aumento dos dados, o modelo deve ter um grande potencial para encontrar similaridade semântica. Os *embeddings* foram gerados via *Word2Vec* e a *tokenização* feita por uma biblioteca em *Python* para língua tailandesa, a *PyThaiNLP*.

Hosseini-Asl e Guha (2015) propõem um modelo de reconhecimento de texto baseado na medição da similaridade visual do texto e na previsão do conteúdo de textos não rotulados. O modelo utiliza uma rede convolucional supervisionada profunda para projetar textos em um espaço de similaridade. Em seguida, um classificador *k-vizinhos* mais próximos é usado para prever o texto não rotulado com base na distância de similaridade para textos rotulados.

O modelo é avaliado em três conjuntos de dados de texto impresso e escrito à mão. Os resultados mostram que o modelo reduz o custo e supera as redes siamesas convencionais. O modelo também é capaz de encontrar textos visualmente semelhantes, mas de difícil leitura, como textos impressos por máquina e manuscritos, devido à supervisão profunda. Além disso, os rótulos previstos pelo modelo às vezes são melhores do que os rótulos humanos, como correção ortográfica.

Rocha e Carvalho (2021) propõem um método baseado em redes siamesas para detecção de relatórios de defeitos duplicados. Uma nova função de perda é apresentada neste trabalho, a *Quintet Loss* que utiliza os centroides dos positivos e dos negativos, além do âncora, do positivo e do negativo normalmente usados na *Triplet Loss*.

A rede siamesa é usada para representar os relatórios no espaço latente, com o modelo BERT, usando a função de perda para aproximar os relatórios duplicados, e além disso utiliza-se modelo de tópicos para extrair metadados afim de melhorar a classificação final de duplicado ou não. Os resultados finais mostraram-se melhores que os trabalhos antes realizados usando as bases *Eclipse, NetBeans e OpenOffice.*

3.2 Busca Semântica e Similaridade Textual

Neculoiu, Versteegh e Rotaru (2016) combinam modelo bi-LSTM com a arquitetura de redes siamesas para aprender a similaridade entre dois textos. O autor utiliza a *Constrative Loss* como função de perda e a distância de cosseno como métrica de similaridade das duas frases.

Os resultados foram comparados com um *baseline* baseado em n-gramas. Os resultados foram bons, porém o autor sugere uma melhor amostra de negativos para o treinamento das redes siamesas.

Feifei, Shuting e Yu (2020) sugerem o uso de Redes Siamesas baseada em BERT para detecção de similaridade semântica de textos. O artigo apresenta uma nova abordagem para correspondência de texto em processamento de linguagem natural, utilizando redes Siamesas e BERT para melhorar a representação semântica entre textos. Ele discute as vantagens dessas técnicas em relação a outras abordagens tradicionais de correspondência de texto e como elas podem ser aplicadas em diferentes áreas.

Viji e Revathy (2022) apresenta uma nova abordagem híbrida para determinar a similaridade de texto usando técnicas de processamento de linguagem natural. A abordagem combina extração de recursos com um BERT com modelo baseada em Redes Siamesas *Bi-LSTM*.

O estudo revisa modelos existentes de detecção de similaridade de texto e explora a arquitetura baseada em BERT para detecção de similaridade de pares de texto, e com isso percebe-se que a seleção das amostras melhora o desempenho em relação às redes neurais em conjuntos de dados em inglês.

O estudo conclui que a abordagem proposta supera os métodos convencionais de detecção de similaridade de texto e pode ser aplicada a outros conjuntos de dados. A arquitetura utilizada no nosso trabalho é bastante similar ao que é mostrado no artigo acima, com a diferença residindo na arquitetura da rede usada: no nosso trabalho uma BERT baseada em *Transformer* e no artigo uma *Bi-LSTM* e a distância para cálculo da similaridade, que no nosso trabalho é Distância de Cosseno e no artigo é a Distância Euclidiana.

Ai et al. (2023) conduz uma pesquisa que discute a interseção entre a área de

Recuperação de Informação (RI) e *Large Language Model* (LLM). A pesquisa destaca a importância dos modelos LLM na recuperação de informação e como eles podem ser usados para melhorar a compreensão do usuário, a geração de texto e a inferência de conhecimento.

Alguns pontos como melhora de RI com LLM são discutidos no artigo, principalmente a melhora de busca da informações. Por fim, são discutidos os desafios como custo computacional, credibilidade das informações geradas e considerações éticas.

3.3 Aumento de Dados

Bonifacio et al. (2022) apresenta uma técnica de aumento de dados chamada *InPars* para melhorar o desempenho de modelos neurais em tarefas de Recuperação de Informações (IR) usando modelos de linguagem pré-treinados. A técnica usa a capacidade de aprendizado com poucas amostras dos LLMs pré-treinados como geradores de dados sintéticos.

O artigo apresenta resultados experimentais que mostram que a técnica *InPars* supera os métodos de recuperação e os *baselines* em várias tarefas de RI. Foi utilizada a *GPT*-3 para a geração dos dados sintéticos, e o MS MARCO *dataset* para avaliar a técnica proposta.

Uma segunda versão do *InPars*, publicada no artigo Jeronymo et al. (2023), os autores agora utilizam um *reranker* como um mecanismo de filtro dos melhores dados sintéticos gerados pelo LLM para assim melhorar as avaliações.

Outra diferença é o modelo utilizado, na primeira versão foi utilizado a *GPT-3* uma API proprietária, nesse artigo é utilizada a *GPT-J*, de código aberto. Para selecionar os melhores dados sintéticos, é usado um modelo *monot5-3b* com *fine-tuning* para *rankear* os melhores dados sintéticos para serem usados.

3.4 Automação de Teste de Software

Esnaashari e Damia (2021) apresentam uma abordagem inovadora para a geração automatizada de dados de teste de *software*. Os autores utilizam algoritmos genéticos e aprendizado por reforço para gerar dados de teste estruturais, reduzindo o custo e o tempo de teste de *software*.

O trabalho apresenta o problema da geração de dados de teste e como a automação pode ajudar a solucioná-lo. Além disso, o artigo apresenta resultados experimentais que mostram a eficácia da abordagem proposta. A abordagem proposta é a geração automatizada de dados de teste de *software*, utilizando algoritmos genéticos e aprendizado por reforço.

A abordagem consiste em duas etapas principais: a primeira etapa é a geração de uma população inicial de dados de teste, utilizando algoritmos genéticos. A segunda etapa é a seleção dos dados de teste mais adequados, utilizando aprendizado por reforço. Os resultados mostraram que a abordagem proposta é capaz de gerar dados de teste estruturais com alta cobertura de código e baixo custo computacional.

Amaricai e Constantinescu (2014) discutem estruturas de automação de testes, expondo vantagens e desvantagens. Além disso propõe uma estrutura de automação parecida com a estrutura que utilizamos neste trabalho, ou seja, reúne um conjunto de comandos que implementam um caso de teste para evitar repetição de teste. Neste trabalho também, eles utilizam palavras chaves para identificar os casos de teste para posterior busca.

Alguns trabalhos realizam uma pesquisa mais abrangente sobre a automação de teste de software, tais como Umar e Zhanfang (2019), Sneha e Malle (2017), Battina (2019) e Rafi et al. (2012), que são trabalhos que realizam uma revisão sistemática sobre o assunto.

Winkler et al. (2010) também propõem um *framework* para automação de casos de teste, baseado em modelo UML, aplicando-o na indústria. O objetivo é melhorar a qualidade do produto e o processo de desenvolvimento de sistemas de automação.

O *framework* permite a geração automática de casos de teste a partir de modelos de sistema, o que economiza tempo e esforço na criação manual de casos de teste. Ele

também fornece uma abordagem sistemática para a execução de casos de teste em um ambiente de teste controlado e a geração de relatórios de teste automatizados, o que ajuda a melhorar a eficiência e a qualidade dos testes em sistemas de automação.

3.5 Síntese dos Trabalhos Relacionados

A Tabela 1 mostra um resumo dos trabalhos relacionados apresentados nesta seção.

Autor	Objetivo	Modelos	Resumo
Rocha e Carvalho (2021)	Similaridade Textual	BERT, MLP, LDA	Detecção de relatórios de defeitos duplicados utilizando redes siamesas com uma função de perda personalizada *(Quintet Loss)* para fazer *clustering* dos relatórios similares.
Poksappaiboon et al. (2021)	Similaridade Textual	MaLSTM	Busca Semântica com Redes Siamesas, usando LSTM com distância de *Manhantan* e *Embeddings* gerados via *Word2Vec* para *chats* de conversa.
Feifei, Shuting e Yu (2020)	Similaridade Textual	BERT, RoBERTa	Busca Semântica com Redes Siamesas, usando BERT, incluindo duas camadas no fim da rede uma de concatenação e outra de predição com a *HingeLoss*.
Viji e Revathy (2022)	Busca Semântica	BERT, bi-LSTM	Busca Semântica com Redes Siamesas, usando BERT pré-treinada para extração de caraterísticas e uma rede *bi-LSTM* para treinamento da rede de siamesa de similaridade para *dataset* de conversas (*chats*).
Bonifacio et al. (2022)	Aumento de Dados	GPT-3	Aumento de dados utilizando a GPT-3, usando *prompts* para geração de dados sintéticos.
Jeronymo et al. (2023)	Aumento de Dados	GPT-J	Evolução do artigo anterior (BONIFACIO et al., 2022) agora com um LLM de código aberto (GPT-J) e filtrando os dados sintéticos gerados.
Este Estudo	**Aumento de Dados e Busca Semântica**	**RoBERTa, BLOOM**	**Busca Semântica com Redes Siamesas, com aumento de dados e fusão de *ranking*. Base de dados própria.**

Tabela 1 – Síntese dos trabalhos relacionados

A Tabela 1 acima mostra que trabalhos como Feifei, Shuting e Yu (2020) e Viji e Revathy (2022) utilizam a ideia de redes siamesas usando modelos baseados em *Transformers* para vetorização dos dados.

São trabalhos bem similares, utilizando o modelo BERT com foco na busca semântica. E o trabalho de Rocha e Carvalho (2021) tem um foco de encontrar documentos similaridades (relatórios de defeitos) utilizando também a ideia de redes siamesas.

Já os trabalhos Bonifacio et al. (2022) e Jeronymo et al. (2023) tem um foco maior no aumento de dados, especificamente usando LLM's para isso. A ideia geral dos dois trabalhos é utilizar o conhecimento das LLM's para, com *prompts* bem ajustados, gerar dados sintéticos semelhantes aos dados reais.

Os dois trabalhos se completam, tendo em vista que os autores colaboraram nos dois trabalhos, o mais recente Jeronymo et al. (2023) utiliza LLM de código aberto ao contrário do primeiro, e utiliza filtros para selecionar os melhores dados sintéticos gerados.

Esta dissertação tem o objetivo maior de realizar a busca semântica com dados de domínios distintos e além disso, com poucos exemplos. Então houve a combinação das duas ideias gerais propostas pelos trabalhos relacionados citados acima, a busca semântica utilizando o modelo *RoBERTa* com aumento de dados utilizando *prompts* para o modelo BLOOM.

4

SOLUÇÃO PROPOSTA

4.1 Descrição do Problema

O presente trabalho usa como base de estudo um *framework* de automação proprietário adotado por uma empresa multinacional, utilizado para a manutenção e execução dos casos de teste, onde há comandos pré-programados que executam determinadas ações de teste em dispositivos *Android*.

O *framework* possui diversos comandos implementados, porém menos de 10% deles tem um passo associado, ou seja, ainda não foram utilizados para automatizar nenhum caso de teste. Isso deve-se ao fato de que é necessária uma mão-de-obra especializada a fim de fazer a associação de cada passo para o comando que executa o teste alvo. Apesar de se utilizar este *framework* neste trabalho, as conclusões e técnicas aqui descritas podem ser utilizadas em outros cenários similares.

Conforme dito na Seção 1, um **passo de teste** é um passo específico (escrito em linguagem natural) que descreve a ação a ser executada no contexto de teste de software, e um conjunto desses passos é denominado **caso de teste**. Já os **comandos de automação** são funções (escritas em alguma linguagem de programação) que executam as ações descritas nos passos de teste. Para ilustrar melhor esse problema, apresenta-se abaixo dois exemplos de passos e seus respectivos comandos correspondentes:

Passo de Teste: *"launch chrome"*

Comando: *"chrome.main.navigate"*

Passo de Teste: *"open quick settings"*

Comando: *"statusbar.navigate"*

Como pode ser visto, tanto passos quanto os comandos nesse cenário são trechos de textos geralmente curtos, que podem ou não ter termos comuns entre si. Além disso, os comandos geralmente vêm em conjuntos de itens muito semelhantes: comandos relacionados ao uso do navegador *Chrome* por exemplo, como atualizar uma página, abrir um *link* em uma nova guia e outras ações podem ter comandos muito semelhantes, compartilhando o prefixo `chrome.main.*`, o que poderia tornar o processo de recuperação muito mais difícil, especialmente em um cenário com mais de 30 mil comandos, como é o nosso caso. Também são comuns casos em que diferentes passos correspondem à um único comando, como pode ser visto na Tabela 2 abaixo:

Passo	**Comando**
launch work camera app	*camera.navigate*
launch personal camera	*camera.navigate*
open camera by quick draw shortcut	*camera.navigate*
open camera app	*camera.navigate*
open the application	*camera.navigate*
open camera	*camera.navigate*
launch camera	*camera.navigate*
open an app that not supports picture - in - picture mode	*camera.navigate*
open an app that not supports picture in picture mode	*camera.navigate*
open the app which launches the permission review screen	*camera.navigate*
open apps that don ' t support multi - window	*camera.navigate*
launch camera	*camera.navigate*
open the camera app	*camera.navigate*
launch personal camera	*camera.navigate*
launch camera app and capture images in	*camera.navigate*
open app and capture photos	*camera.navigate*

Tabela 2 – Exemplos de um comando de automação com vários passos de teste associados

Nesse caso exemplificado na Tabela 2 acima, mostra-se alguns passos diferentes que apontam para um mesmo comando. Em um cenário de busca, isso acrescenta dificuldade pois não há um padrão do tipo de passo que um determinado comando está relacionado.

No cenário de automação de casos de teste, neste estudo a intenção é ajudar os programadores a encontrar o comando correto para automatizar cada passo. Para isso, (1) é necessário recuperar uma lista de **comandos candidatos** que provavelmente automatizarão o referido **passo de teste** e (2) usar filtros com base nos metadados do

caso de teste, métodos de extração de entidades e recursos não textuais para podar a lista de candidatos.

Neste trabalho, interessa-se as técnicas para melhorar a geração de candidatos (1), uma vez que (2) é fortemente baseado em características muito específicas da estrutura de teste do *framework* utilizado e não deve fornecer *insights* generalizáveis para leitores interessados em outros cenários.

Também propõe-se o uso de um método de **fusão de *ranking*** para unificar estas duas listas de candidatos, que, em nos experimentos deste trabalho, obteve os melhores resultados. Foi utilizado o algoritmo **LambdaMART** para unir os *rankings* gerados pelos mecanismos de busca desenvolvidos.

A figura 9 ilustra a arquitetura geral proposta para esse estudo.

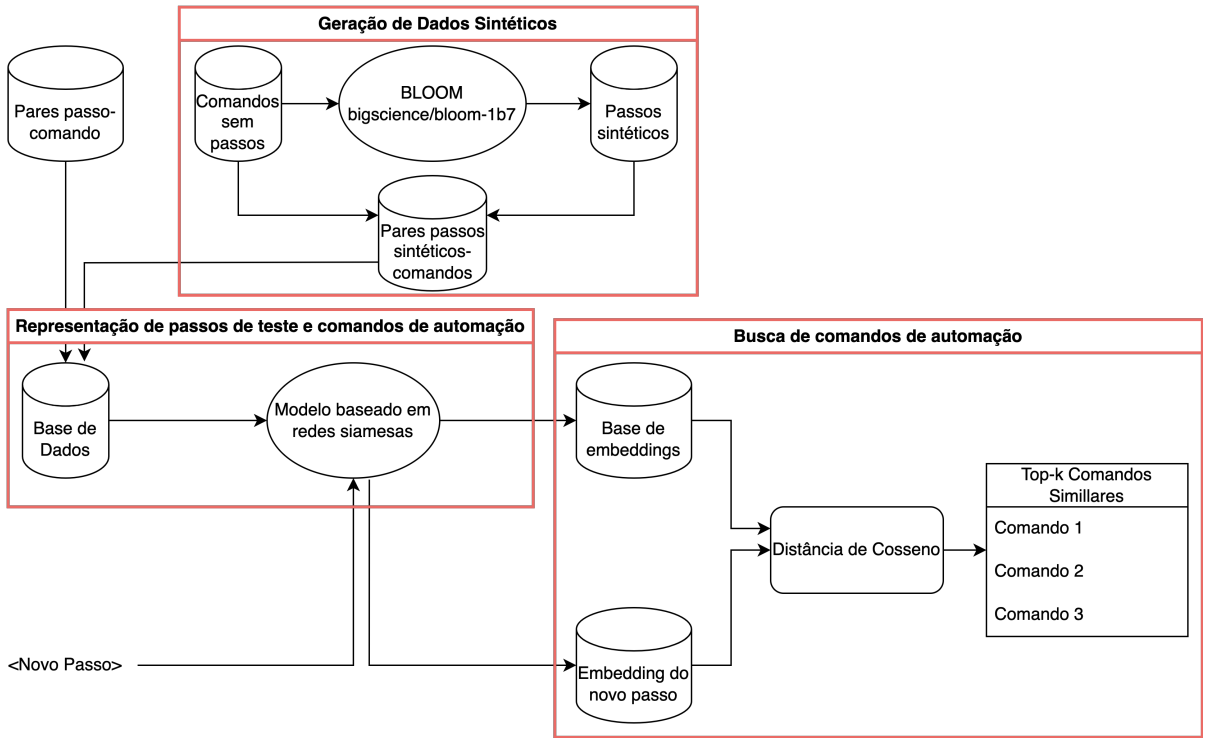

Figura 9 – Arquitetura Geral do Método Proposto

Fonte: Próprio Autor

A Figura 9 acima, mostra que a arquitetura é composta por três grandes etapas (destacadas em vermelho):

- **Representação de Passos de Teste e Comandos de Automação**: Essa etapa en-

volve o treinamento das redes siamesas para representação dos dois tipos de dados utilizados: os passos de teste e os comandos de automação;

- **Geração de Dados Sintéticos**: Como há comandos sem passo de teste associado, essa etapa é responsável por gerar passos sintéticos para esses comandos;
- **Busca de Comandos de Automação**: É a etapa de busca dos comandos de automação dado um passo de teste. Essa etapa também envolve fusão de ranking para melhorar a lista de comandos possíveis.

Essa etapas serão explicadas com mais detalhamento nas próximas seções.

4.2 Representação de passos e comandos

Para representar tanto os passos de teste quanto os comandos de automação, propõe-se o uso de uma combinação de MLM (*Masked Language Model*) e Redes Siamesas. A Figura 10 apresenta a arquitetura geral da rede neural utilizada para representação dos passos e dos comandos:

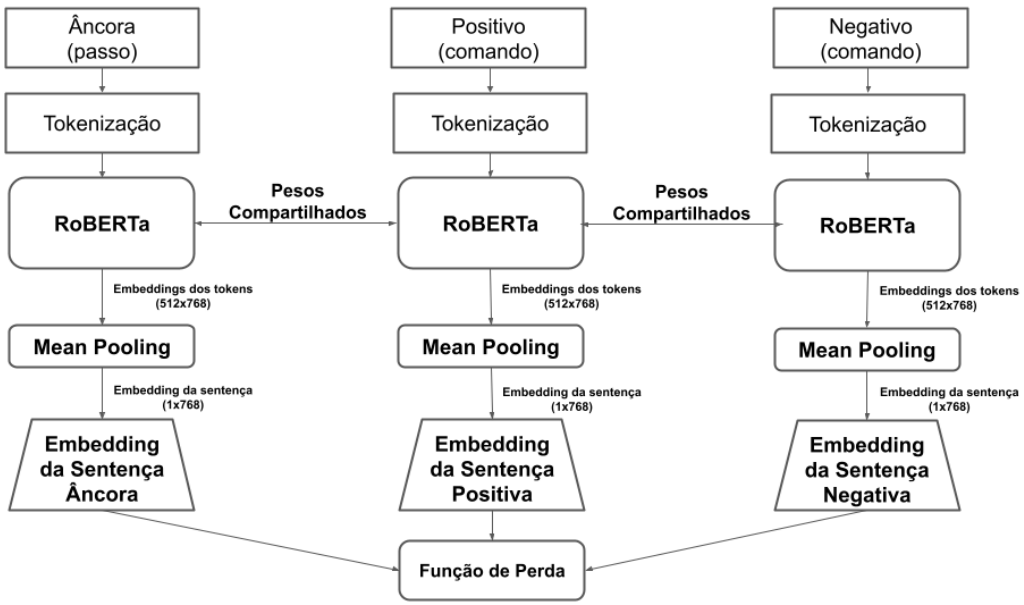

Figura 10 – Arquitetura do modelo de redes siamesas para representação de passos e comandos

Fonte: Próprio Autor

Para representar as características textuais, dos passos e dos comandos, foi utilizado o *RoBERTa* (DEVLIN et al., 2018), uma rede neural pré-treinada baseada em *Transformer* comumente utilizada em diversas tarefas de PLN. *RoBERTa* é um modelo *BERT* (DEVLIN et al., 2018) treinado com um conjunto de dados diferente e com uma otimização de hiperparâmetros mais robusta do que o modelo *BERT* original.

A arquitetura geral do método proposto é uma Rede Siamesa (CHICCO, 2021), uma arquitetura onde, ao ter duas ou mais ramificações de entrada da rede compartilhando pesos, a rede pode representar diferentes entradas no mesmo espaço latente e, dependendo de sua função de perda, pode ser comumente usada em tarefas de similaridade de itens.

Ao utilizar uma Rede Siamesa, a intenção é representar tanto os comandos quanto os passos no mesmo espaço latente. Primeiro os dados são *tokenizados* e passados para o modelo *RoBERTa* para o treinamento do mesmo e após isso o modelo gera *embeddings* dos *tokens* dados como entrada. Com os *embeddings* dos *tokens*, foram feitos um *mean pooling* baseado na média dos vetores dos *tokens* para assim gerar o *embedding* da frase.

Para usar a *Triplet Loss*, a função de perda deste trabalho, um passo é sempre considerado o âncora, o seu comando correspondente é usado como exemplo positivo e um outro comando que não seja correspondente é usado como exemplo negativo.

Assim, espera-se que a distância entre a representação vetorial de um passo e o seu comando correspondente seja menor do que a de outros comandos, já que a *Triplet-Loss* minimiza a distância entre o âncora e o positivo e maximiza a distância entre o âncora e o negativo, tornando a busca pelo comando correspondente uma tarefa simples de recuperação *top-k.* Então, para esta função de perda, primeiramente foi utilizada a versão *Semi-Hard* da *Triplet Loss* para escolher o melhor exemplo negativo.

A Figura 11 mostra a diferença entre exemplos negativos "*hard*" (exemplos negativos que estão no raio entre o âncora e o positivo), "*semi-hard*" (exemplos negativos que estão no raio entre o positivo e um margem especificada como parâmetro) e "*easy*" (exemplos negativos que estão no raio fora da margem). A semi-hard foca nos exemplos negativos que estão próximos do exemplo positivo, para diminuir o erro entre os grupos

de comandos formados ao fim do treinamento.

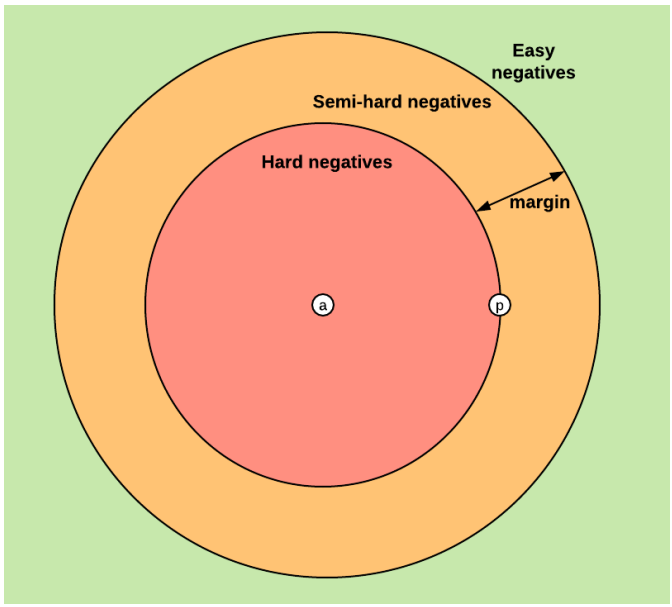

Figura 11 – Exemplos de tipos de negativos

Fonte: Moindrot (2018)

A Figura 12 exemplifica o que acontece no treinamento das redes siamesas com a Triplet-Loss com os nossos dados: no espaço vetorial da esquerda, o passos ”open camera app” e ”launch personal camera” e o comando associado à eles ”camera.navigate” estão representados um longe do outro no espaço vetorial. Após o treinamento, os vetores dos mesmos ficam próximos e quando o passo ”launch personal camera” é representado, ele está próximo do comando ”camera.navigate” ao qual ele é associado.

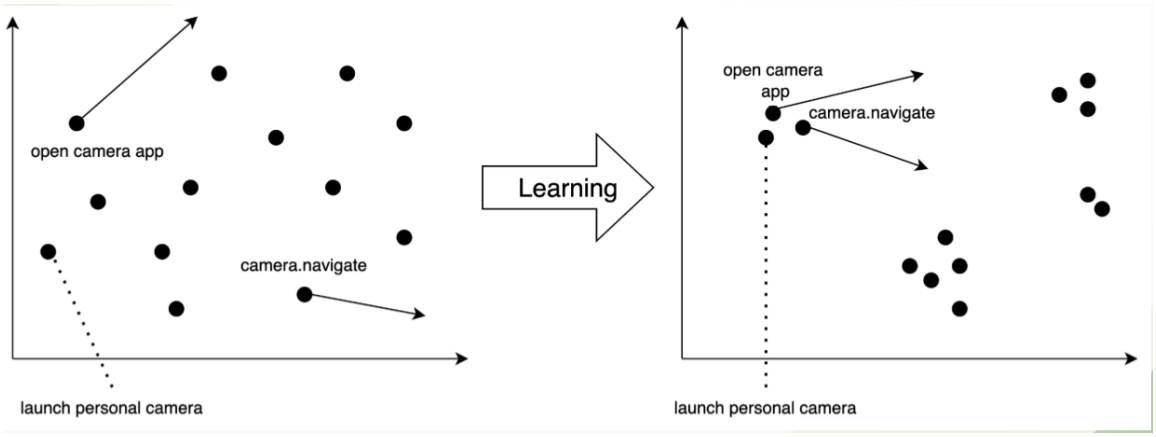

Figura 12 – Exemplo da *Triplet-Loss* no espaço vetorial de passos e comandos

Fonte: Próprio Autor

Nesse contexto, o âncora sempre vai ser um passo de teste, e o positivo é o comando em que o passo tem relação (o comando que ele é associado) e o negativo um comando não relacionado *Semi-Hard* escolhido aleatoriamente.

Como a implementação de redes siamesas pode **criar representações tanto para comandos quanto para passos**, a mesma rede pode ser utilizada para gerar representações para sistemas de busca, tanto para os comandos diretamente quanto pelas representação dos passos anteriormente atribuídos como automatizáveis por tal comando no processo de teste.

Tal abordagem de correspondência passo a passo deve contornar problemas decorrentes de passos de teste e comandos de automação pertencentes a diferentes domínios. No entanto, isso leva a um dilema, já que muitos dos comandos da nossa base não tem um passo atribuído a ele.

Além disso, em produção, a **grande maioria** dos comandos não tem um passo associado, assim, seria inviável usar a busca passo a passo para esse problema.

Para contornar isso, propõe-se a **geração sintética de passos de teste associados**, descritos na próxima Seção 4.3. A Figura 13, ilustra a arquitetura geral de representação de passos e comandos da base original e da base sintética criada.

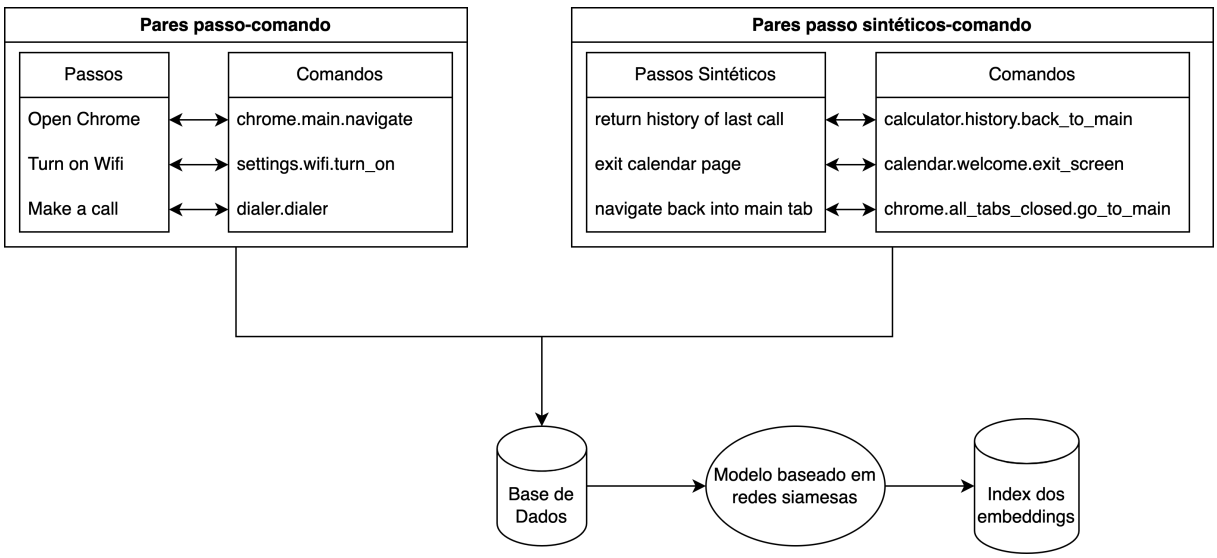

Figura 13 – Arquitetura da representação de passos e comandos

Fonte: Próprio Autor

4.2.0.1 Pré-processando comandos

Os comandos de automação no cenário proposto são basicamente *strings* de acesso a métodos de objetos, sendo assim formadas por sequências de caracteres separadas por '.' (ponto).

Para auxiliar o processo de representação das mesmas em um mesmo espaço de características dos passos de teste, os mesmos são processados e reescritos para se assemelharem mais a linguagem natural. Abaixo, mostra-se um exemplo desta reescrita usando o comando **chrome.main.open_new_tab** de exemplo:

- Quebra o comando pelo caractere "." (ponto) = *[chrome, main, open_new_tab]*;
- Inverte a lista gerada = *[open_new_tab, main, chrome]*;
- Faz uma nova quebra, agora pelo caractere "_" (*underline*) = *[open, new, tab, main, chrome]*;
- Converte em texto (Resultado Final) = ***open new tab main chrome***.

Devido ao fato do último elemento do nome do comando ser o nome de um método, a versão final do comando reescrita torna-se uma frase rudimentar no imperativo, com o primeiro elemento sendo uma espécie de verbo.

Com esse algoritmo, converte-se todos os comandos da base para um texto que se parece mais com uma escrita natural, com o objetivo de que na hora do treinamento da rede siamesas (especificamente na *RoBERTa*, baseada em *BERT*) os *tokens* sejam gerados com a maior representatividade possível.

Se os comandos fossem vetorizados com a escrita original, cada um deles seria um único *token* pois os mesmos não tem separação por espaço e não tem informação semântica (como verbos e entidades), ao contrário dos comandos padronizados com o nosso algoritmo que tem uma similaridade sintática maior com os passos associados a eles.

Também é importante salientar que foram realizados diversos testes com outros modelos *Transformers* até que se chegasse ao modelo atualmente em uso, o *RoBERTa*. Alguns dos modelos que utilizamos na fase de testes foram o *BERT* (e suas variações

quanto ao tamanho do modelo: *small*, *medium* e *large*), o *distilbert* e as variações de tamanho do modelo do *RoBERTa*.

Com o modelo *RoBERTa* utilizado nos experimentos finais, foram gerados seis combinações da nossa abordagem, **sendo três abordagens para cada experimento**, como explicado nas próximas subseções.

4.2.1 Abordagem *Origin*

A primeira abordagem, denominada **Origin**, utiliza o conjunto de dados original para o treinamento sem a inclusão de dados sintéticos. Em seguida, os *embeddings* do conjunto de dados original e os *embeddings* dos comandos do conjunto de teste são gerados para serem incluídos no espaço de busca (o *Index*).

No teste, para cada passo, é gerado o vetor de *embedding* de todos os comandos e seus respectivos passos, e após isso são recuperados k vetores mais semelhantes no espaço de busca de acordo com a distância de cosseno. A Figura 14 ilustra quais bases são usadas no Treino e no *Index* da abordagem **Origin**.

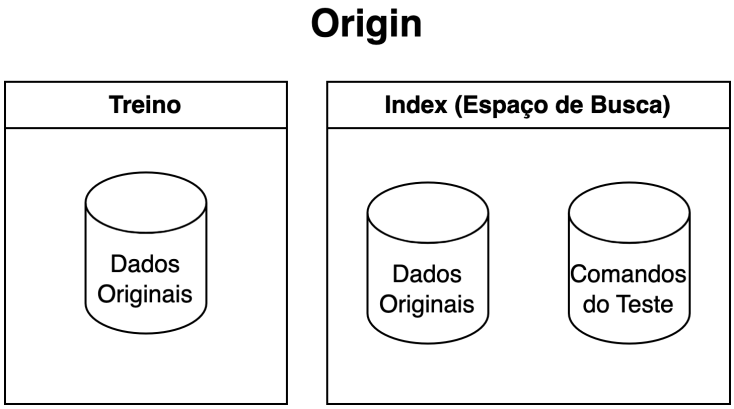

Figura 14 – Abordagem 1 - Origin

Fonte: Próprio Autor

4.2.2 Abordagem *SynTest*

Em uma segunda abordagem, chamada **SynTest**, leva-se em consideração que alguns comandos não tem um par de passo associado e outros só tem um passo associado dificultando a busca por esses comandos.

Assim, representações sintéticas de passos são geradas a fim de aumentar os exemplos que temos, e esses passos são usados indexados para busca. Com isso, os *embeddings* do conjunto de dados original, os *embeddings* dos comandos do conjunto de teste e os *embeddings* dos dados sintéticos são gerados para serem incluídos no espaço de busca (o Índice).

No teste, similar a abordagem anterior (**Origin**), para cada passo, é gerado o vetor de *embedding* e após isso é calculada a similaridade via distância de cosseno com os k vetores mais semelhantes no espaço de busca. A Figura 15 ilustra quais bases são usadas no Treino e no *Index* da abordagem **SynTest**.

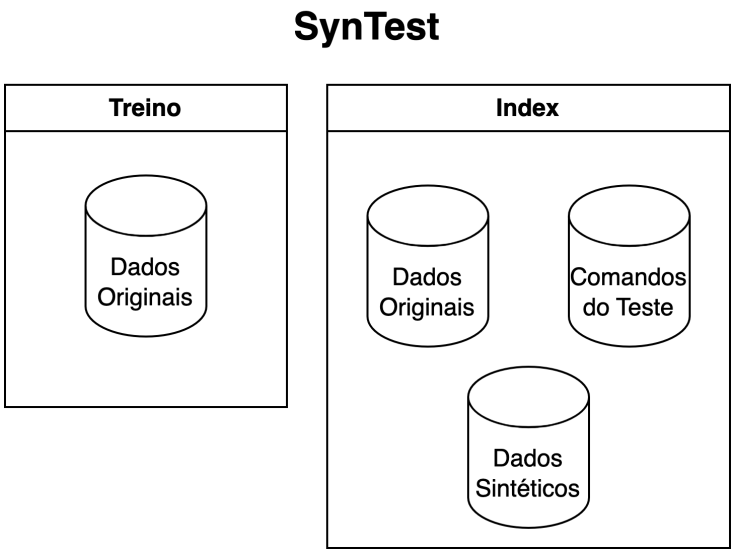

Figura 15 – Abordagem 2 - SynTest

Fonte: Próprio Autor

4.2.3 Abordagem *SynAll*

Na última abordagem,**SynAll**, o modelo treina as redes siamesas usando tanto o conjunto de dados original quanto os dados sintéticos. O espaço de busca (*Index*) e o teste é o mesmo da abordagem anterior (**SynTest**). A Figura 16 ilustra quais bases são usadas no Treino e no *Index* da abordagem **SynAll**.

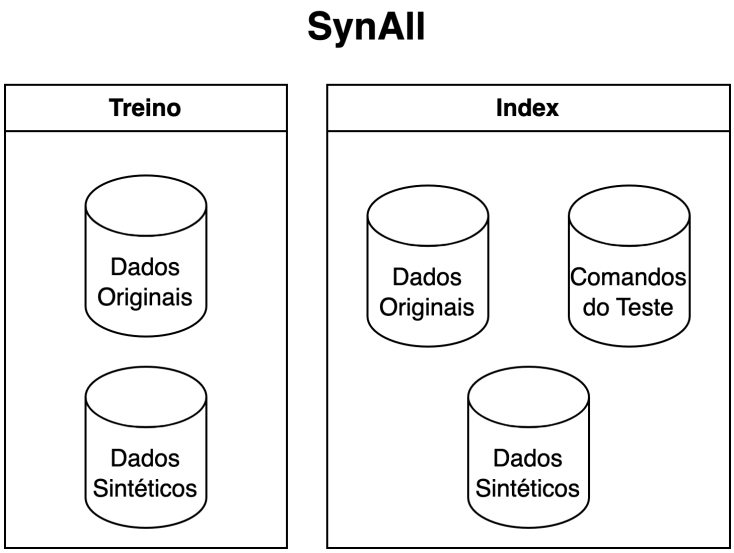

Figura 16 – Abordagem 3 - SynAll

Fonte: Próprio Autor

A próxima Seção, explica com mais detalhes a proposta para geração de dados sintéticos neste estudo.

4.3 Geração de dados sintéticos

Para contornar o problema de relacionar passos de teste em linguagem natural com comandos de automação baseados em linguagem de programação, pode ser interessante uma abordagem baseada na correspondência entre pares de passos , em que um comando é representado pela correspondência de passos de teste que foram previamente identificados como automatizados por um dado comando.

Embora testes preliminares tenham mostrado que essa abordagem tem potencial (muito mais do que outros elementos textuais relacionados aos comandos, como

comentários de código-fonte em sua implementação).

Em cenários com poucos casos de teste previamente automatizados, o número de comandos com passos de teste associados a eles é reduzido, diminuindo radicalmente o impacto desta abordagem, especialmente em termos de revocação.

Assim, para realisticamente utilizar essa abordagem, seria necessário ou anotar manualmente cada comando com o seu passo associado ou implementar algum método para gerar automaticamente passos associados aos comandos. Desta forma, propõe-se a realizar o segundo.

Para gerar esses passos sintéticos usamos um LLM baseado em *Transformers*, o BLOOM (SCAO et al., 2022). O BLOOM é um modelo treinado para geração de texto a partir de um *prompt* em vastas quantidades de dados de texto.

Este modelo é capaz de gerar texto coerente e difícil de distinguir do texto escrito por humanos em 46 idiomas e 13 linguagens de programação

O BLOOM também pode ser instruído a executar tarefas de texto para as quais não foi explicitamente treinado, executando-as como tarefas de geração de texto, utilizando aprendizado baseado em poucas amostras.

Este modelo é uma arquitetura apenas de decodificador (como o GPT-3), sendo que o BLOOM original tem aproximadamente 176 bilhões de parâmetros. Neste trabalho foi utilizado o *bloom-1b7*[1], que é uma pequena versão do BLOOM original e foi treinado com aproximadamente 1,7 bilhões de parâmetros, 24 camadas, 16 cabeças de atenção e 2048 de tamanho de *tokens* de entrada.

Para gerar novos passos, foi utilizado uma abordagem de aprendizado em poucas amostras (WANG et al., 2020; PARNAMI; LEE, 2022). O aprendizado em poucas amostras é um *framework* de meta-aprendizagem em que um modelo grande pré-treinado é usado para uma tarefa de aprendizado de máquina, sendo apresentado a um número muito reduzido de exemplos de treinamento.

Essa abordagem geralmente é usada para se referir a tarefas em que os exemplos são apresentados ao modelo como uma entrada *prompt*, sem ajuste fino de hiperparâmetros realizado no domínio específico.

[1] https://huggingface.co/bigscience/bloom-1b7

Neste caso, apresenta-se o BLOOM com um número de comandos e seus passos associados e solicitá-la para que gere um *passo sintético* para um novo comando, para o qual não há um passo associado.

Para cada comando, é gerado um *prompt* com 10 pares de **passo-comando** mais semelhantes (por similaridade de cosseno) ao que queremos gerar. Abaixo, é apresentada uma amostra de um *prompt* de entrada:

```
"Translate this command to a description

statusbar.expanded.is_rotation_on <EOS> - enable auto rotate button

settings.system.date_time.set_auto_time_off <EOS>
- disable use network - provided time

settings.system.date_time.set_auto_timezone_off <EOS>
- disable use network - provided time zone

actions.actions.swipe_shrink.navigate <EOS>
- disable microscreen

settings.system.date_time.set_auto_timezone_off <EOS>
- disable use network - provided time and use network -
provided time zone on date & time settings

launcher3.app_tray.disable_work_mode <EOS> - disable work profile

settings.security.toggle_one_lock <EOS> - disable one lock

display.peek_display.turn_aod_off <EOS> - disable peek display

system.disable_dark_theme <EOS> - disable dark theme

settings.vpn.turn_off_always_on <EOS> - disable always on vpn

statusbar.expanded.turn_rotation_off <EOS> - <PASSO_PREDITO>"
```

Com o *prompt* de exemplo apresentado acima, a saída do modelo é um passo sintético **(<PASSO_PREDITO>)** que o modelo sugere como um passo associado para o comando dado como exemplo (**statusbar.expanded.turn_rotation_off**) baseados nos dez exemplos de pares passo-comando dados como entrada.

Assim, o modelo ”aprende” com esses exemplos e gera dados sintéticos. Utilizando essa abordagem, consegue-se ampliar a base de dados rotulando de forma sintética os comandos que não continham passos correspondentes. A Figura 17, ilustra a arquitetura geral de geração de dados sintéticos.

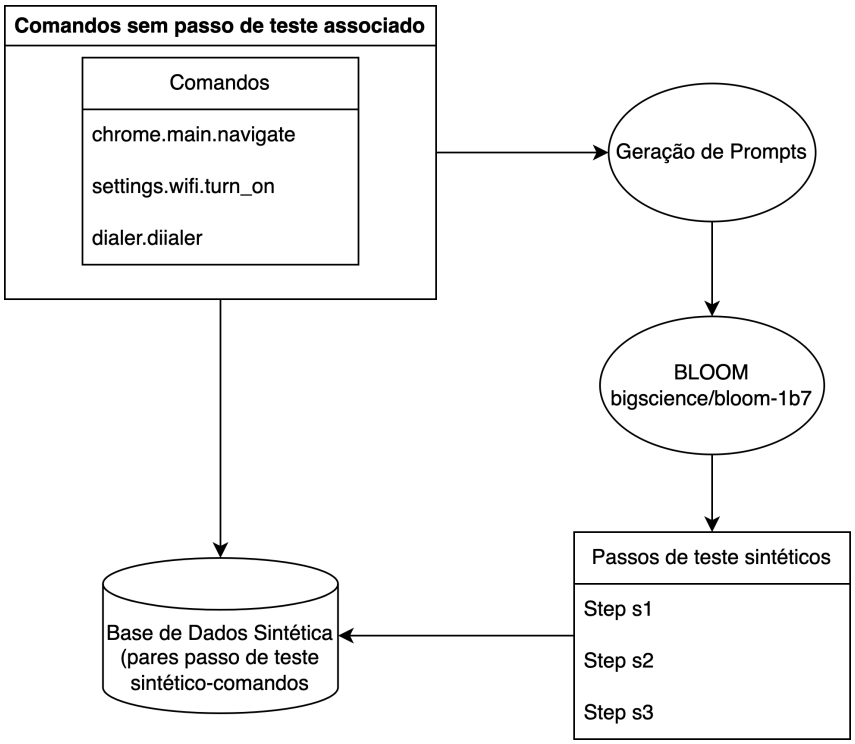

Figura 17 – Arquitetura da geração de dados sintéticos

Fonte: Próprio Autor

Para criação do *prompt*, foram recuperados os **dez** comandos mais similares ao comando no qual queremos gerar um passo sintético, por distância de cosseno. A partir disso é gerado o texto do *prompt* em um padrão pré-estabelecido e por fim o modelo gera um passo sintético que associamos com o comando. A Figura 18 explica os passos descritos abaixo.

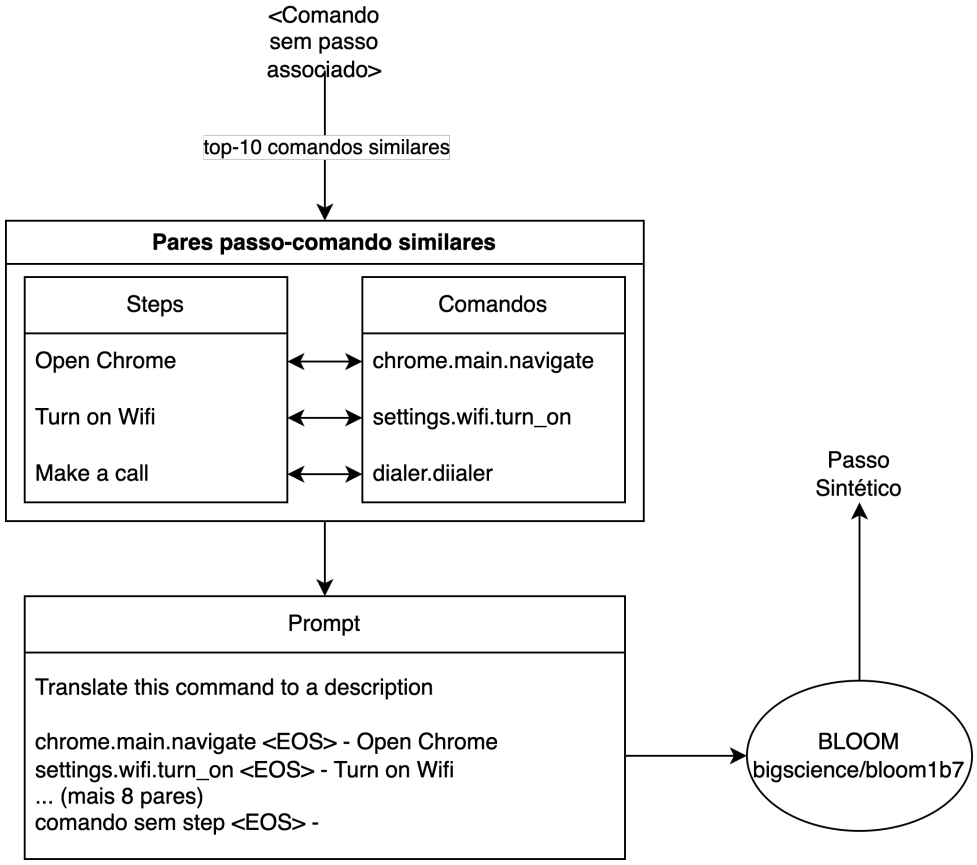

Figura 18 – Diagrama da geração de dados sintéticos

Fonte: Próprio Autor

O *framework* utilizado no nosso estudo, contém mais de 30000 comandos de automação, sendo que a grande maioria não contém um passo de teste associado. Este trabalho, utilizou uma amostra desses comandos, coletando principalmente os comandos que tem passos de teste associados, ou seja, comandos que já foram automatizados.

Dito isso, o *dataset* utilizado neste estudo, têm ao todo 1.529 comandos únicos, e todos esses comandos tem ao menos um passo associado. Utilizando o método de geração de dados sintéticos, foram criados dois passos sintéticos para cada comando da base de dados a fim de ajudar o modelo a representar os passos que têm relação com um comando.

Ao todo, 3.058 novos passos sintéticos foram criados. A Tabela 3 abaixo mostra alguns exemplos dos passos criados a partir de alguns comandos, e uma avaliação qualitativa dessa geração sintética.

Passo sintético	Comando original	Avaliação
return history of last call	calculator.history.back_to_main	Bom
is in history view or not ?	calculator.history.is_in_screen	Ruim
multiply numbers by 10, 100 etc.	calculator.main.multiply	Bom
sum up all numbers in your calculator	calculator.main.sum	Bom
clear calendar view (only for Android)	calendar.clear_calendar	Bom
back from search results page	calendar.event.back_to_search	Bom
answer incoming calls (if available)	calling.answer_call	Bom
wait for active calls in background	calling.wait_for_active_call	Bom
wait for no calls in last 30 seconds	calling.wait_for_no_call	Bom

Tabela 3 – Análise qualitativa de alguns passos de teste sintéticos gerados pelo modelo BLOOM

No geral, o modelo gerou passos sintéticos bem condizentes com o esperado e bem parecido com os passos reais. Foi realizada uma análise qualitativa com uma amostra de cem passos gerados, e desses cem foi avaliado que apenas cinco não estavam bem escritos pelo modelo, ou seja, os passos gerados não pareciam com passos anteriormente escrito por operadores humanos e em alguns casos, o modelo gerou passos em que haviam muitas informações irrelevantes para o objetivo do passo de teste.

4.4 Busca de Comandos de Automação

Para buscar o comando de automação correto para automatizar um passo de teste, propõe-se três métodos de busca diferentes:

- **Busca *Passo-Passo* (1)** : Em vez de comandos, são geradas representações para os passos de teste que foram previamente encontrados para automatizar os referidos comandos. No caso de comandos que não têm nenhum passo de teste associado, pode-se (1) utilizar passos de teste sintéticos para representá-los ou (2) simplesmente ignorá-los, em variações que não usam geração sintética. Nos experimentos, este método é identificado pelo prefixo `sstep`. A Figura 19, mostra como é feita a busca passo-passo.

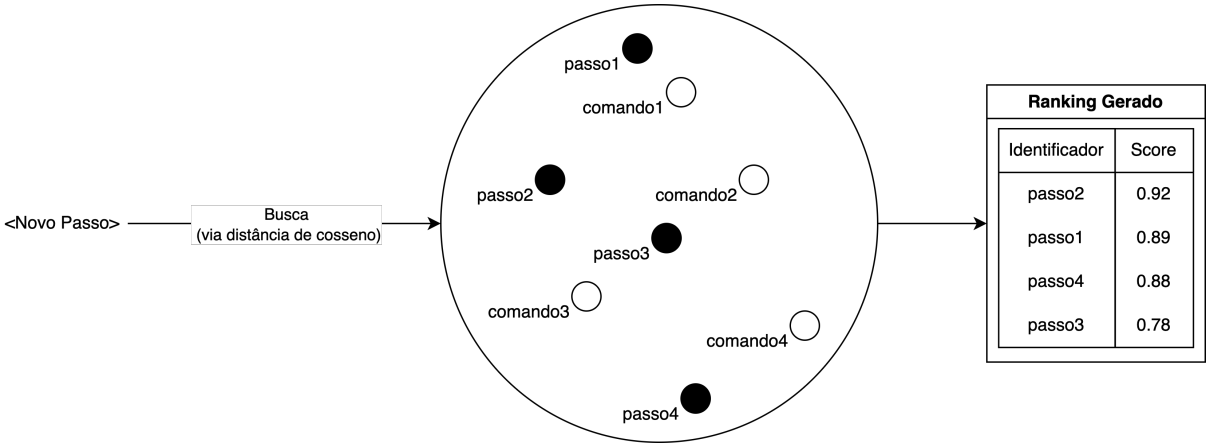

Figura 19 – Busca Passo-Passo

Fonte: Próprio Autor

- **Busca *Passo-Comando* (2)** : Representações vetoriais de todos os comandos são geradas e indexadas, e sempre que há um novo passo de teste a ser automatizado, sua representação é gerada pelo mesmo conjunto de pesos treinados, e os k comandos mais similares são recuperados. Nos experimentos, este método é identificado pelo prefixo `scom`. A Figura 20, mostra como é feita a busca passo-comando.

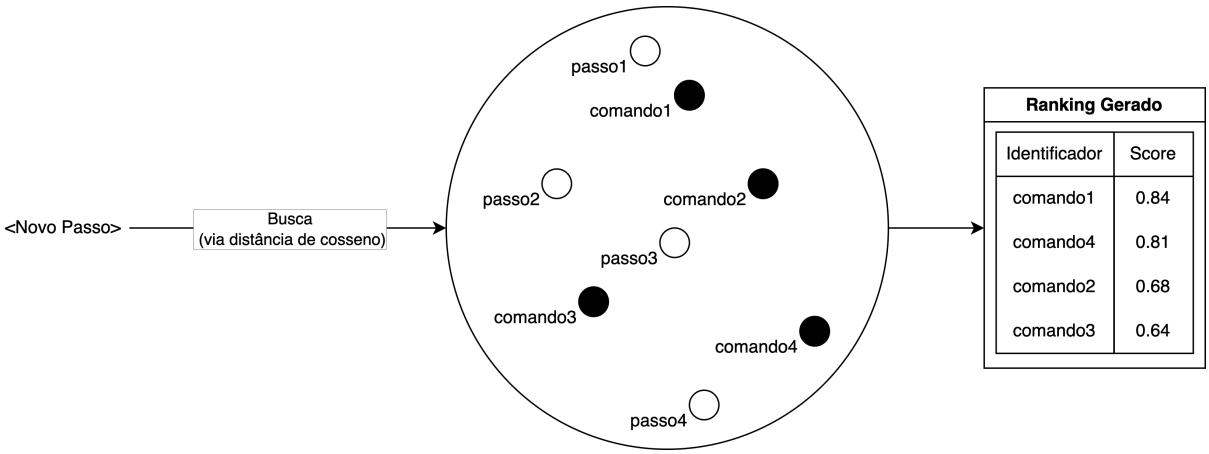

Figura 20 – Busca Passo-Comando

Fonte: Próprio Autor

- **Busca *Passo-(Passo+Comando)* (3)** : Nesta busca, tanto os passos quanto os comandos são indexados em um único índice. Desta forma, a busca considera tanto comandos diretamente quanto passos associados a comandos.

  Nos experimentos, este método é identificado pelo prefixo `sstepcom`. A Figura 21, mostra como é feita a busca passo-passo+comando.

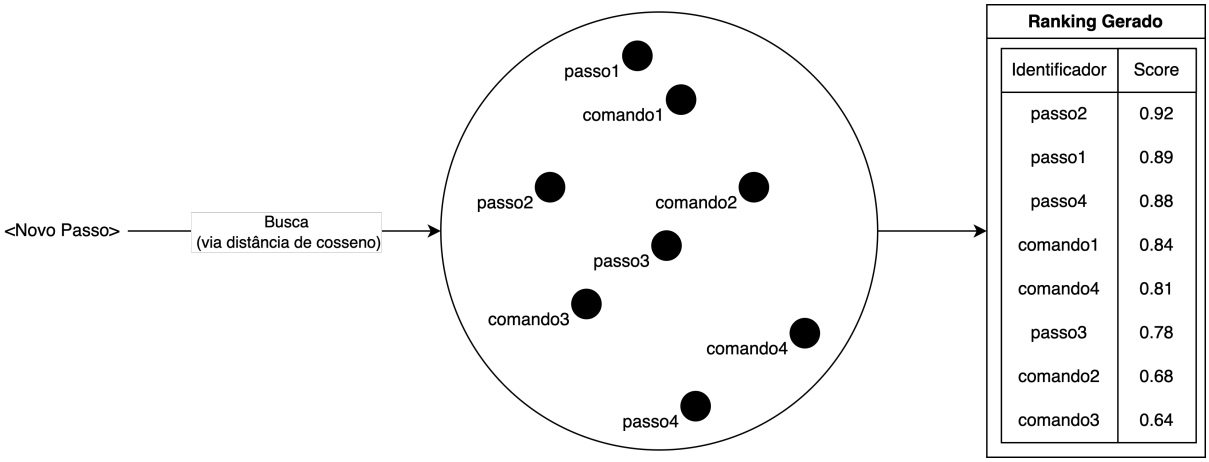

Figura 21 – Busca Passo-Passo+Comando

Fonte: Próprio Autor

Além disso, foram utilizadas técnicas de **Fusão de *Rank*** para unificar as três buscas em uma única classificação. Neste trabalho, utilizou-se o **LambdaMART** (BURGES, 2010; LI; LIU, 2018) para a fusão dos *rankings*. Nos experimentos, este método é identificado pelo prefixo `fusion`.

Nesse caso, foram criadas quatro fusões (combinando as três buscas): **fusion12**, **fusion13**, **fusion23** e a fusão das três buscas juntas **fusion123**. Os números correspondem às buscas originais: **sstep** (1), **scom** (2) e **sstepcom** (3). Exemplificando: fusion23 quer dizer a fusão dos *rankings* **scom** (2) e **sstepcom** (3).

Em relação a criação (ou não) de passos sintéticos associados a comandos, têm-se três cenários possíveis, conforme explicitado na Seção 4.2:

- **Conjunto de dados original**. Nenhuma ampliação é feita. Comandos que não possuem um passo associado conhecido não podem ser recuperados pela busca. Este cenário é identificado nos experimentos pelo sufixo `Origin`.

- **Dados sintéticos no *index***. Foram criados passos de teste sintéticos para todos os comandos, e os *embeddings* desse passos foram gerados pelo modelo treinado somente com os dados original, ou seja, usa-se o modelo que não foi treinado com dados sintéticos para representar os passos de teste sintéticos e indexá-los. Este cenário é identificado nos experimentos pelo sufixo `SynTest`.

- **Dados sintéticos no treino e no *index***. Foram criados passos de teste sintéticos para todos os comandos, e esse dados foram treinados no modelo juntamente com os dados originais. Após isso todos os comandos e passos são indexados para a busca. Este cenário é identificado nos experimentos pelo sufixo `SynAll`.

A Figura 22, ilustra a arquitetura geral da busca de comandos de automação. Os retângulos destacados em negrito são os diferentes *rankings* de busca que foram utilizados.

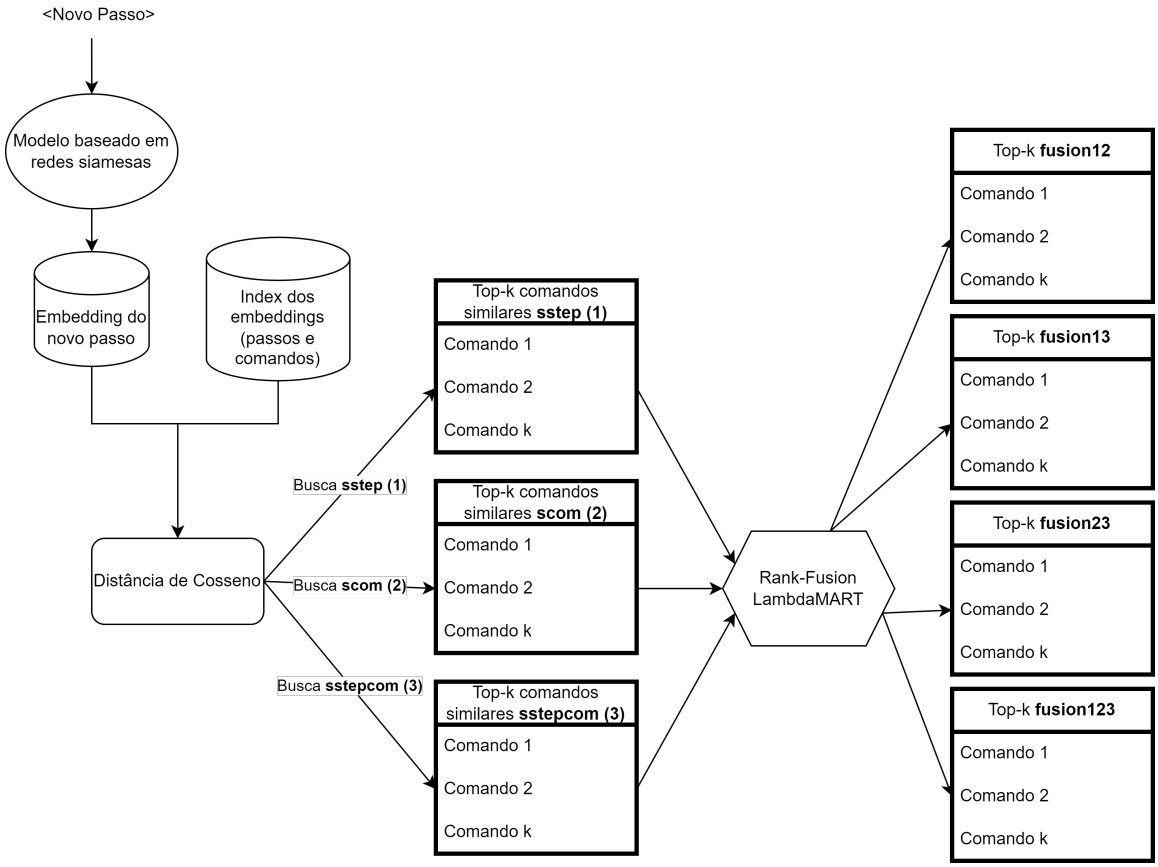

Figura 22 – Arquitetura geral da busca de comandos de automação

Fonte: Próprio Autor

4.5 Considerações Finais

Conforme explicado no decorrer desta Seção, o método proposto consiste em combinações do modelo de representação baseado em redes siamesas treinado com diferentes conjuntos de dados, o **Origin** onde o treinamento e o *Index* não contém os dados sintéticos, o **SynTest** que utilizando os dados originais para treinamento e os dados sintéticos no *Index* e o **SynAll** que utiliza os dados sintéticos no treinamento e no *Index*.

Além disso, foi realizado também três tipos de busca **(Origin, SynTest e SynAll)** e técnicas de fusão de *rank* para fundi-las, gerando assim mais quatro resultados (combinando as três buscas).

Para cada modelo apresentado, o treinamento é feito utilizando a função de perda e além disso, dois experimentos distintos são feitos conforme explicado na Seção 5.1, utilizado os modelos apresentados. Com isso, gerando seis combinações das abordagens. Abaixo, apresenta-se um resumo dos experimentos:

- Experimentos: **Exp1 e Exp2**;
- Abordagens de representação: **Origin, SynTest e SynAll**;
- Buscas: **sstep, scom, sstepcom, fusion12, fusion13, fusion23 e fusion123**;

Na próxima Seção, mostra-se os resultados finais dos experimentos realizados.

5

EXPERIMENTOS

Neste capítulo, apresenta-se os experimentos realizados utilizando o método para busca de Comandos de Automação dado um Passo de Teste de entrada.

Apresenta-se também os modelos e hiperparâmetros utilizados e os resultados finais separados pelos dois experimentos propostos.

As próximas Seções estão dispostas da seguinte forma: na Seção 5.1 explica-se os dados utilizados nos experimentos, bem como estatísticas sobre os mesmos; na Seção 5.2 explica-se como foi gerado o *baseline* para comparação com o modelo proposto; na Seção 5.3 explica-se os hiperparâmetros de cada modelo utilizado nos neste experimento; na Seções 5.4 e 5.5 mostra-se os resultados finais da busca no conjunto de dados Exp1 e Exp2; e por fim , na Seção 5.6 conclui-se o capítulo resumindo os resultados apresentados.

5.1 Conjunto de Dados de Treinamento

Para realizar os experimentos, foram usados um conjunto de dados rotulados manualmente, contendo 5.795 pares de comandos de automação e seus respectivos passos de teste associados, extraídos de aproximadamente 1.229 casos de teste do uso anterior do *framework* utilizado nesse trabalho.

Nessa base de dados, foram obtidos 1.529 comandos de automação únicos e 4.159 passos de teste únicos. Esses pares foram divididos em três conjuntos: o de Treino, o de Teste e o de Validação.

Para os experimentos, a divisão da base em Treino, Teste e Validação foi feita de duas formas distintas:

- **Divisão por Passos de Teste únicos**: Nessa divisão, a base de Treino, Teste e Validação foram criadas da seguinte forma: 50% dos **passos únicos** (e seus pares de comandos associados) foram incluídos na base de Treino; 20% foram incluídos na base de Validação; e por fim, os 30% restantes foram incluídos na base de Teste.

  Os experimentos realizados com essa configuração de base de dados são identificados pelo prefixo **`Exp1`**. O diagrama exemplificando essa divisão pode ser visto na Figura 23.

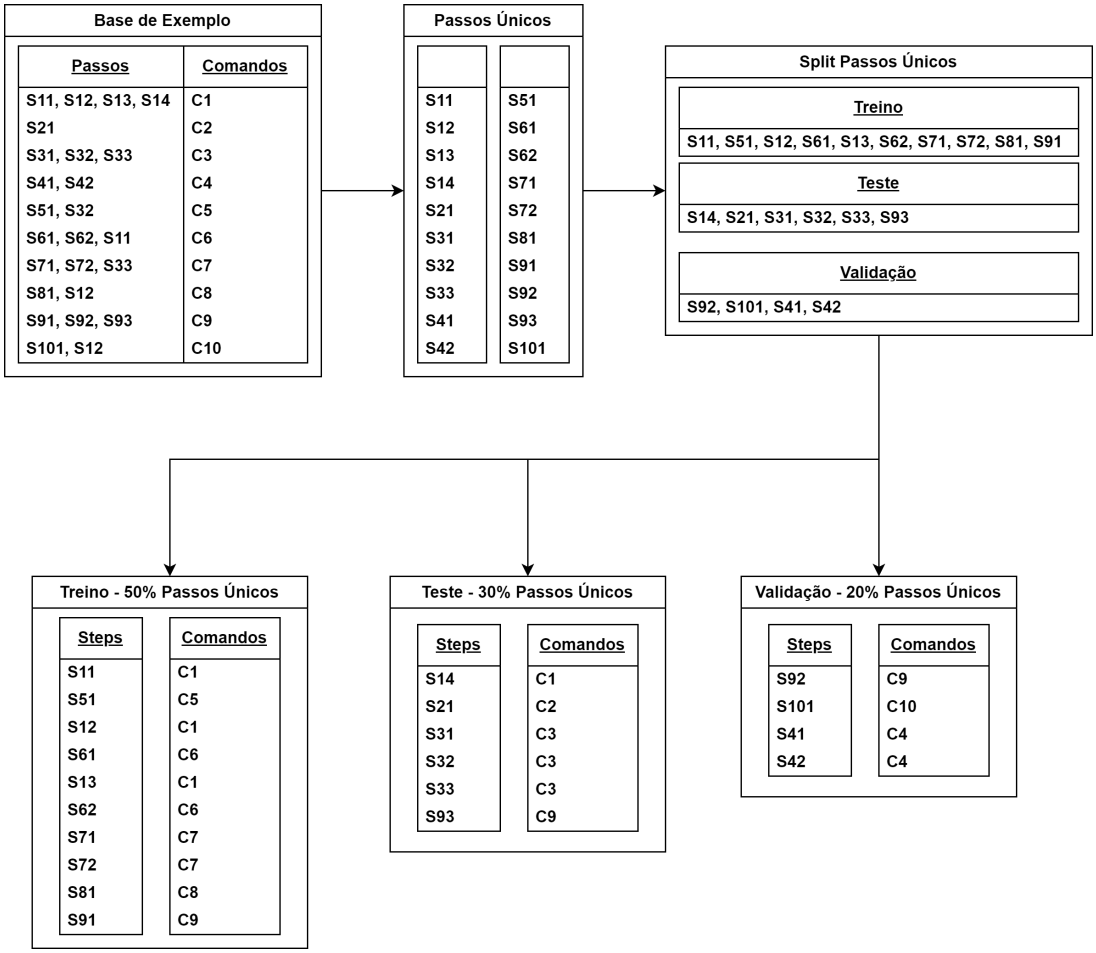

Figura 23 – Diagrama de Divisão dos dados do Exp1

Fonte: Próprio Autor

- **Divisão por Comandos de Automação únicos**: Seguindo a mesma lógica da divisão do **Exp1**, 50% dos **comandos únicos** (e seus pares de passos de teste associados) foram incluídos na base de Treino; 20% foram incluídos na base de Validação; e por fim, os 30% restantes foram incluídos na base de Teste.

  Os experimentos realizados com essa configuração de base de dados são identificados pelo prefixo **`Exp2`**. O diagrama exemplificando essa divisão pode ser visto na Figura 24.

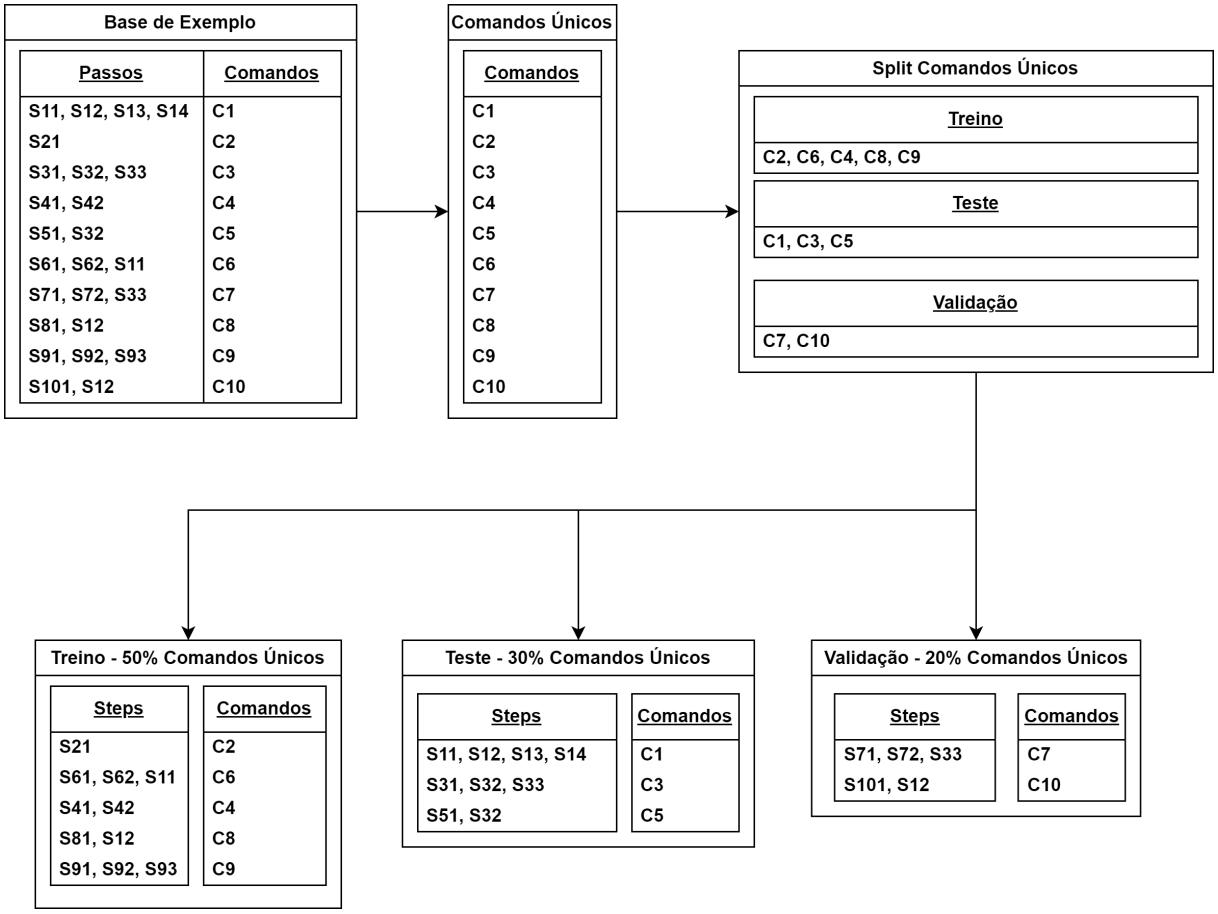

Figura 24 – Diagrama de Divisão dos dados do Exp2

Fonte: Próprio Autor

Essa divisão em dois experimentos ocorre porque de fato temos os dois cenários em produção: O Exp2, que é a divisão por comandos, simula o que atualmente temos na base de dados que são comandos sem um par de passo de teste associado.

Porém, com o uso contínuo da ferramenta de automação, a ideia é que do Exp2 possamos partir para o Exp1, em que a maioria dos comandos já tem um par de passo de teste associado, tornando-se cada vez mais raro comandos não automatizados.

A Tabela 4, mostra as estatísticas finais dos dois *datasets* gerados:

Experimentos	Exp1			Exp2		
Divisão	Treino	Teste	Validação	Treino	Teste	Validação
Nº de Pares Passo-Comando	3247	1762	786	2989	1608	1198
Nº Passos Únicos	2328	1248	583	2479	1418	1097
Nº Comandos Únicos	1125	832	476	856	459	214
Nº de Passos do Teste contidos no Treino	-	0	0	-	430	303
Nº de Comandos do Teste contidos no Treino	-	542	329	-	0	0

Tabela 4 – Estatísticas dos Dados dos dois experimentos

Explicando os dados contidos na Tabela 4 acima, temos as seguintes informações:

- **Experimentos**: Mostra os experimentos propostos, o Exp1 e o Exp2;
- **Divisão**: Mostra as três divisões da base feitas para a execução dos experimentos: Treino: para o treinamento do modelo; Teste: para avaliar o quão bom foi o modelo; e Validação: também para avaliar o quão bom foi o modelo (já que a validação é bem parecida com o Teste), porém a principal função da validação no nosso trabalho é gerar a base de treinamento do algoritmo de fusão *LambdaMART*;
- **Nº de Pares Passo-Comando**: Quantidade de pares únicos Passo de Teste-Comando de Automação contido nas bases de Treino, Teste e Validação;
- **Nº Passos Únicos**: Quantidade de Passo de Teste únicos contidos nas três bases: Treino, Teste e Validação. Os valores variam um pouco em comparação com a quantidade de pares Passo-Comando pois alguns passos de teste tem mais de um comando associado e alguns comandos tem mais de um passo de teste associado;
- **Nº Comandos Únicos**: Quantidade de Comandos de Automação únicos contidos nas três bases: Treino, Teste e Validação. Os valores também variam, pois alguns passos de teste tem mais de um comando associado e alguns comandos tem mais de um passo de teste associado;
- **Nº de Passos do Teste contidos no Treino**: Quantidade de Passos que estão no Teste e também estão no Treino. Essa informação é importante pois conforme a nossa divisão, é importante saber se no Exp1, por exemplo, em que a divisão é feita por passos quanto comandos estão no Treino;

- **Nº de Comandos do Teste contidos no Treino**: Quantidade de Comandos de Automação que estão no Teste e também estão no Treino. Essa informação é importante, pois conforme a divisão, é importante saber se no Exp2, por exemplo, em que a divisão é feita por comandos quanto Passos estão no Treino;

Conforme a Tabela 4, para cada experimento têm-se a quantidade de dados diferentes, tendo em vista que alguns passo tem mais de um comando associado e o inverso também acontece, pois alguns comandos tem mais de um passo associado.

Uma coisa a se destacar é que quando faz-se a divisão por passos únicos, tendo em vista que alguns comandos tem mais de um passo associado, têm-se comandos que estarão no Treino e no Teste (com dois passos diferentes associados). A mesma coisa se aplica na divisão por comandos únicos, no qual há passos de teste que estarão no Treino e no Teste.

Com essas estatísticas, também pode-se ver a razão pela qual é necessário os dados sintéticos nos dois experimentos. No Exp1, no teste, há 832 comandos únicos dos quais 542 estão representados no teste com um passo de teste real associados.

Os outros 290 comandos, nunca poderão ser buscados via busca **sstep**, pois os mesmos não tem um passo de teste associado, e para isso foi incluso os passos de teste sintéticos, que ajudaram principalmente nos comandos em que não há pares associados.

No Exp2 que esse problema se destaca, tendo em vista que nenhum dos comandos do Teste estão no Treino, não podendo os mesmos serem encontrados via busca **sstep** necessitando ainda mais dos dados sintéticos.

5.2 *Baseline*

Para melhor entendermos o impacto que as abordagens propostas têm na solução do problema, se faz necessário utilizar algum método base para comparação.

Após testes preliminares com diversos modelos clássicos de recuperação de informação, como modelo vetorial e BM25 (BAEZA-YATES; RIBEIRO-NETO, 2013), que não obtiveram bons resultados, foi decidido pela utilização de duas abordagens para gerar os resultados iniciais baseadas em busca de *embeddings*, usando o modelo de linguagem RoBERTa com *Fine-Tuning*, para gerar os *embeddings* dos nossos dados e assim realizar as buscas.

Este modelo foi escolhido por ser o mesmo que foi utilizado como parte da abordagem com redes siamesas. Desta forma, o diferencial entre seus resultados ajuda a explicar o quão impactante é o método proposto, baseado em redes siamesas e quanto da qualidade advém do modelo de linguagem.

Propondo o uso de duas abordagens, o **BaselineSAUG** e o **BaselineCAUG** como *baseline* de comparação. O **BaselineSAUG** utiliza os dados originais, sem inclusão de dados sintéticos para a busca. Já o **BaselineCAUG** adiciona nossa base sintética ao dados originais no espaço de busca.

O *Fine-Tuning* é feito utilizando os mesmos dados utilizados nos treinos dos dois experimentos (Exp1 e Exp2). Essa etapa foi realizada para que o modelo se adapte aos dados do domínio (caso de teste de *software*) para que assim os *embeddings* gerados sejam os mais representativos possíveis.

5.3 Hiperparâmetros

Conforme explicado na Seção 4.2, foi criado um modelo de Redes Siamesas com o objetivo de representar tanto os comandos quanto os passos no mesmo espaço latente.

Esse modelo foi treinado por 800 épocas, salvando um *checkpoint* a cada 100 épocas para posterior avaliação. Nessa etapa, percebe-se que o treinamento convergiu à medida que chegava na 300º época e após isso apresentava *overfitting*. Então, os modelos passaram a ser treinados somente pelas 300 épocas. Os hiperparâmetros gerais do treinamento são apresentados na Tabela 5.

modelo	épocas	*batch*	*loss*
RoBERTa-base	300	256	*triplet-loss*

Tabela 5 – Hiperparâmetros das Redes Siamesas

Conforme explicado na Seção 5.2, foi feito um *Fine-Tuning* no modelo RoBERTa-base para que assim os *embeddings* gerados pelo modelo Baseline fossem mais representativos, tendo em vista que a adaptação de domínio ajuda nessa tarefa. Os hiperparâmetros gerais do treinamento são apresentados na Tabela 6.

modelo	épocas	*batch*
RoBERTa-base	300	256

Tabela 6 – Hiperparâmetros do *Fine-Tuning* do Baseline

Em relação a geração de dados sintéticos, conforme foi explicado na Seção 4.3, tendo em vista que nossa base de dados tem comandos com somente **um** passo associado ou até mesmo comandos sem **nenhum** passo associado, foi necessário criar dados sintéticos para contornar esse problema.

Para cada comando, foi criado **dois** passos sintéticos para que os mesmos tenham representação de um passo associado na etapa de busca, alguns desses comandos geraram passos iguais, ficando com apenas **um** passo associado.

Para que fossem gerados esses dados, foram treinados *prompts* baseados na base de dados a fim de dar como entrada para um modelo LLM exemplos de como são os padrões de escrita tanto dos comandos quanto dos passos conforme explicado na 4.3.

Para essa geração foi utilizado modelos LLM's que recebem um *prompt* como entrada e gera a continuação desse texto como saída. Neste trabalho, foi utilizado o modelo BLOOM [1] pois, o mesmo foi o que gerou textos mais similares aos resultados esperados. Os hiperparâmetros são mostrados na Tabela 7:

modelo	*max_lenght*	*top_k*	*temperature*
BLOOM	300	1	0.9

Tabela 7 – Hiperparâmetros da Geração de Dados Sintéticos

[1] bigscience/bloom1b7

5.4 Resultados - Exp1

5.4.1 Dados Originais - Exp1

Inicialmente, foi comparado os resultados do *baseline* com dados originais do modelo proposto baseado em redes siamesas também com dados originais (sem a adição de dados sintéticos).

A ideia de mostrar essa comparação é validar se o modelo proposto baseado em redes siamesas realmente ajuda na representação de passos de teste e comandos de automação no mesmo espaço de busca. Além disso, pode-se verificar se a função de perda, que tem o objetivo principal de aproximar os pares de passos e comandos está conseguindo obter sucesso na sua proposta.

A Tabela 8, mostra os dados descritivos da pontuação MRR e MAP, e a Figura 25, mostra o gráfico da pontuação MRR e a Figura 26, mostra o gráfico da pontuação MAP dessa comparação.

MODELO	BUSCA	MRR	MAP
BaselineSAUG	sstep	0.37	0.38
BaselineSAUG	scom	0.14	0.13
BaselineSAUG	sstepcom	0.37	0.39
Origin	sstep	0.49	0.52
Origin	scom	**0.56**	0.59
Origin	sstepcom	**0.56**	**0.60**

Tabela 8 – Valores de MRR e MAP dos modelos com dados originais - Exp1

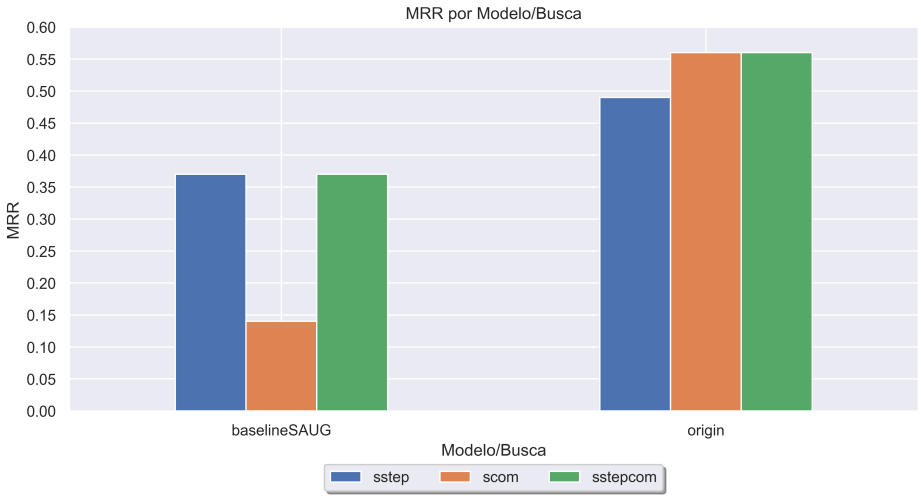

Figura 25 – MRR dos modelos com dados originais - Exp1

Fonte: Próprio Autor

Figura 26 – MAP dos modelos com dados originais - Exp1

Fonte: Próprio Autor

Os resultados mostrados nas Figuras 25 e 26 mostram que a utilização do modelo baseado em redes siamesas realmente ajuda na representação dos passos de teste e dos comandos no mesmo espaço e melhora as pontuações tanto no MRR quanto no MAP. Além disso comparando somente a busca usando *embeddings* de comandos (**scom**), pode-se ver um aumento considerável do *baseline* para o modelo siamês sem aumento de dados (Origin).

Isso valida ainda mais que a representação dos comandos de automação no mesmo espaço do passo de teste ajuda a obtermos os candidatos relevantes nas posições mais altas dos rankings. Este ganho se reflete também quando se representa passos e comandos no mesmo espaço (sstepcom).

Seguindo nessa ideia, também é possível comparar a pontuação *HitRate@n* dos dois modelos. Avaliar a pontuação *HitRate* é importante nesse trabalho, pois ela mostra a capacidade que a busca proposta tem de recuperar pelo menos um comando de automação relevante nas posições mais altas do *ranking* gerado.

No caso deste estudo, avalia-se o *HitRate@n*, onde o *n* é o número de candidatos recuperados, e o *HitRate* variando entre 1 e 0. Exemplificando: *HR@1* avalia se a busca recuperou um comando relevante na primeira posição do *ranking*(se achou a pontuação é 1, senão é 0), *HR@3* avalia se a busca recuperou pelo menos um comando relevante em uma das três primeiras posições do *ranking* (se achou pelo menos um a pontuação é 1, senão é 0), *HR@5* avalia se a busca recuperou pelo menos um comando relevante em uma das cinco primeiras posições do ranking (se achou pelo menos um a pontuação é 1, senão é 0), e assim sucessivamente para os

outros valores de n.

Em comparação às outras métricas usadas na área de Recuperação de Informação (RI), para este trabalho o *HitRate*, o MRR e o MAP são as métricas que melhor se adequaram ao modelo proposto, tendo em vista que no neste caso não foi preciso todos os comandos relevantes nas primeiras posições, bastou encontrar um relevante. Além do mais, a relevância é binária, ou seja, varia entre 0 e 1, não havendo comandos mais relevantes que outros, todos estão no mesmo nível de relevância.

Esta é uma das razões pelas quais não foram utilizados uma das métricas comumente usadas para avaliar mecanismos de busca, que é o *NDCG*.

A Tabela 9, mostra a descrição dos dados e a Figura 27, mostra os dados plotados em um gráfico de linhas para verificar a evolução de cada busca a medida que se aumenta o *top-n* candidatos do *HitRate@n.*

MODELO	BUSCA	HR@1	HR@3	HR@5	HR@10	HR@20	HR@30	HR@40	HR@50
BaselineSAUG	sstep	0.27	0.44	0.48	0.53	0.57	0.60	0.62	0.63
BaselineSAUG	scom	0.08	0.16	0.19	0.26	0.32	0.37	0.41	0.44
BaselineSAUG	sstepcom	0.27	0.45	0.50	0.57	0.61	0.65	0.67	0.68
Origin	sstep	0.40	0.57	0.61	0.66	0.70	0.71	0.72	0.72
Origin	scom	**0.44**	**0.65**	**0.72**	**0.79**	**0.83**	0.84	**0.86**	**0.87**
Origin	sstepcom	**0.44**	**0.65**	**0.72**	0.78	**0.83**	**0.85**	**0.86**	**0.87**

Tabela 9 – Valores de HitRate@n dos modelos com dados originais - Exp1

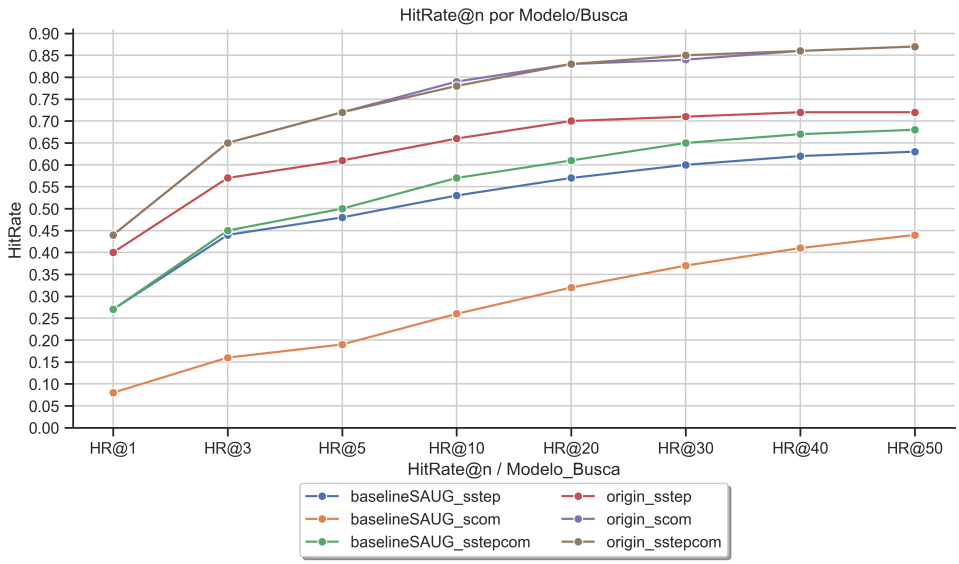

Figura 27 – HitRate@n dos modelos com dados originais - Exp1

Fonte: Próprio Autor

Analisando o gráfico mostrado na Figura 27, pode-se ver que o modelo **Origin** com as duas buscas concatenadas **sstepcom** e a busca por comandos **scom** foi o que gerou o melhor *ranking*, e se manteve como melhores resultados, mesmo aumentando o valor de *top-n*.

Uma característica interessante é que a busca somente por passos **sstep** tem uma diferença de pontuação crescente quando comparada à busca **scom**, a partir do *HR@1*. Isto pode significar que existem alguns casos em que a representação do passo consulta é muito próxima da de um passo correto na base de dados.

Conforme mais posições são consideradas, esta vantagem some e a representação de comandos se mostra vantajosa. Isto se reflete também ao se colocar passos e comandos no mesmo espaço de busca.

No geral, os resultados acima são uma importante indicação que (1) Usar redes siamesas para criar representações de comandos nos mesmos domínios que os passos conseguiu ganhos significativos (2) que estes ganhos também se estendem à representação dos passos.

5.4.2 Dados Originais + Sintéticos - Exp1

Nesta Seção, compara-se o impacto que os dados sintéticos tiveram nesta tarefa de busca. Para uma melhor organização, primeiro foi analisado o impacto dos dados sintéticos no *baseline* e na sequência na rede siamesa proposta.

5.4.2.1 Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados

Inicialmente, foi avaliado esse impacto no modelo usado como *baseline*. Com isso pretende-se validar se a etapa de geração de dados sintéticos ajudou a buscar os comandos que não tinham um par de passo de teste associado.

A Tabela 10, mostra os dados descritivos da pontuação MRR e MAP, a Figura 28, mostra o gráfico da pontuação MRR e a Figura 29, mostra o gráfico da pontuação MAP dessa comparação.

MODELO	BUSCA	MRR	MAP
BaselineSAUG	sstep	0.37	0.38
BaselineSAUG	scom	0.14	0.13
BaselineSAUG	sstepcom	0.37	**0.39**
BaselineCAUG	sstep	**0.38**	**0.39**
BaselineCAUG	scom	0.14	0.13
BaselineCAUG	sstepcom	**0.38**	**0.39**

Tabela 10 – Valores de MRR e MAP - Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados - Exp1

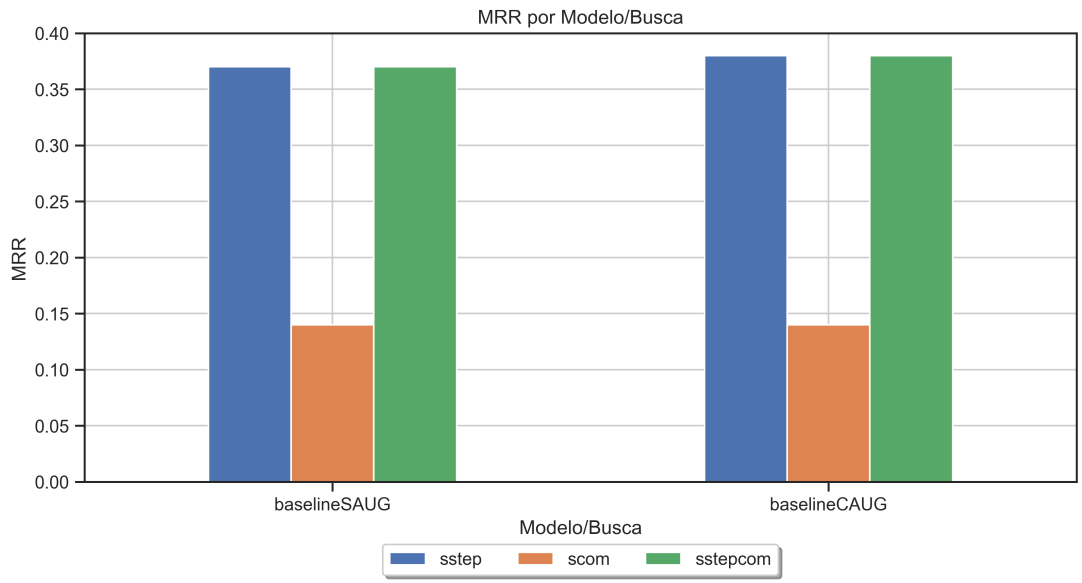

Figura 28 – MRR - Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados - Exp1

Fonte: Próprio Autor

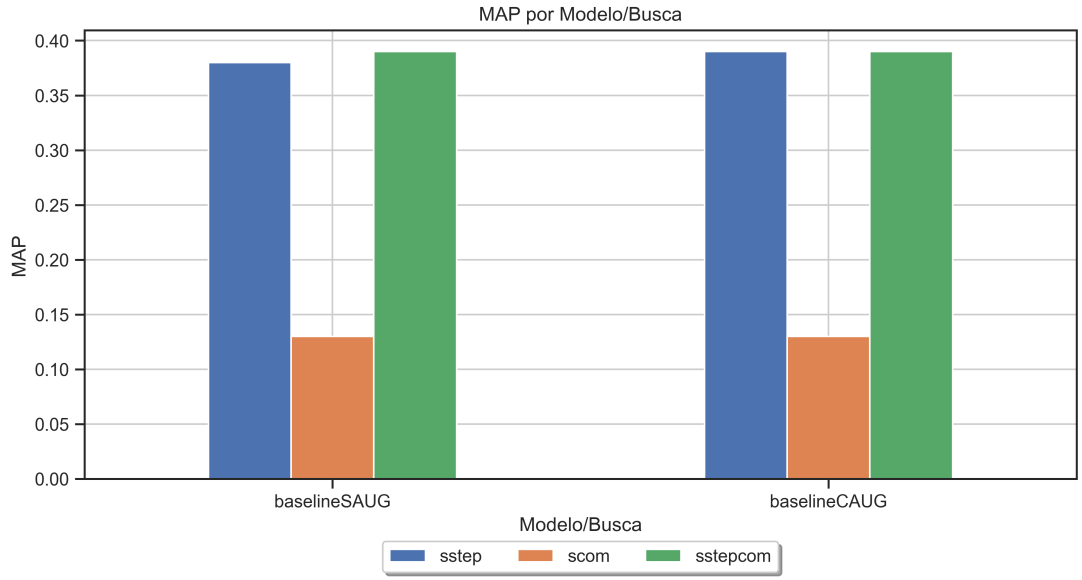

Figura 29 – MAP - Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados - Exp1

Fonte: Próprio Autor

Os resultados mostrados nas Figuras 28 e 29, mostram que a inclusão dos dados sintéticos na busca no *baseline* não impactou de forma significativa os resultados, aumentando bem pouco a pontuação do MRR e do MAP ***(+0.01)*** na busca **sstep** por exemplo.

Seguindo nessa comparação, também compara-se a pontuação dos dois modelos, o *baseline* sem aumento de dados e o *baseline* com aumento de dados.

A Tabela 11, mostra a descrição dos dados e a Figura 30, mostra os dados plotados em um gráfico de linhas para verificar a evolução de cada busca a medida que se aumenta o *top-n* candidatos.

MODELO	BUSCA	HR@1	HR@3	HR@5	HR@10	HR@20	HR@30	HR@40	HR@50
BaselineSAUG	sstep	0.27	0.44	0.48	0.53	0.57	0.60	0.62	0.63
BaselineSAUG	scom	0.08	0.16	0.19	0.26	0.32	0.37	0.41	0.44
BaselineSAUG	sstepcom	0.27	**0.45**	**0.50**	**0.57**	0.61	0.65	0.67	0.68
BaselineCAUG	sstep	**0.28**	**0.45**	0.49	**0.57**	0.63	**0.66**	**0.69**	**0.71**
BaselineCAUG	scom	0.08	0.16	0.19	0.26	0.32	0.37	0.41	0.44
BaselineCAUG	sstepcom	**0.28**	0.44	**0.50**	0.56	**0.64**	**0.66**	**0.69**	**0.71**

Tabela 11 – Valores de HitRate@n - Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados - Exp1

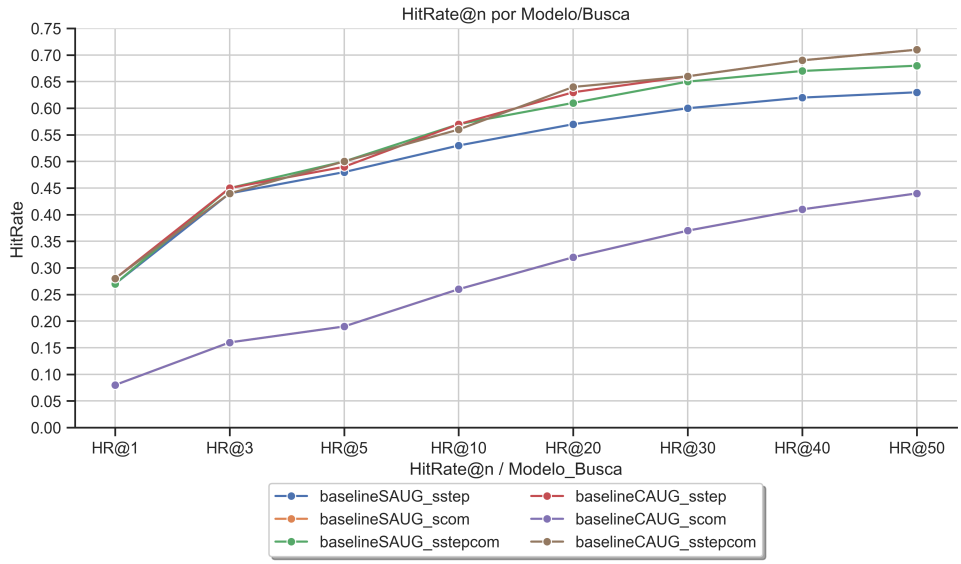

Figura 30 – HitRate@n - Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados - Exp1

Fonte: Próprio Autor

Analisando o gráfico mostrado na Figura 30, pode-se afirmar que os dados sintéticos, de modo geral, ajudam na busca **sstep** e **sstepcom**.

Os melhores resultados gerais foram nos modelos que foram utilizados na busca **sstep-com**.Também é importante observar que, foram comparados somente o *baseline*, a busca **scom** é uma busca ruim tendo em vista que, em termos semânticos, o passo e o comando são de diferentes domínios.

Também é importante salientar que, a inclusão de dados sintéticos não afeta a busca **scom**, sendo que nos dois modelos (sem aumento de dados e com aumento de dados) os valores são idênticos.

Isso deve-se ao fato de que, como a busca **scom** busca diretamente nos comandos, a inclusão de passos de teste sintéticos não as afeta tendo em vista que só há inclusão de passos de teste sintéticos e não de comandos sintéticos.

5.4.3 Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados

Essa etapa, compara o impacto que os dados sintéticos tiveram na tarefa de busca no nosso modelo baseado em redes siamesas. A Tabela 12, mostra os dados descritivos da pontuação MRR e MAP, a Figura 31, mostra o gráfico da pontuação MRR e a Figura 32, mostra o gráfico da pontuação MAP dessa comparação.

MODELO	BUSCA	MRR	MAP
Origin	sstep	0.49	0.52
Origin	scom	0.56	0.59
Origin	sstepcom	0.56	0.60
SynTest	sstep	0.52	0.56
SynTest	scom	0.56	0.59
SynTest	sstepcom	0.56	0.60
SynAll	sstep	0.55	0.59
SynAll	scom	**0.57**	**0.61**
SynAll	sstepcom	**0.57**	0.60

Tabela 12 – Valores de MRR e MAP - Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados - Exp1

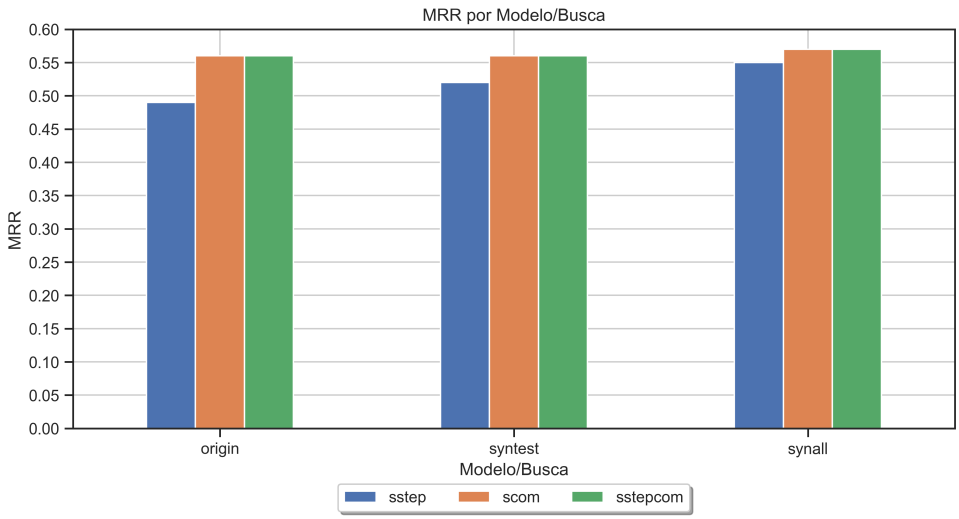

Figura 31 – MRR - Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados - Exp1

Fonte: Próprio Autor

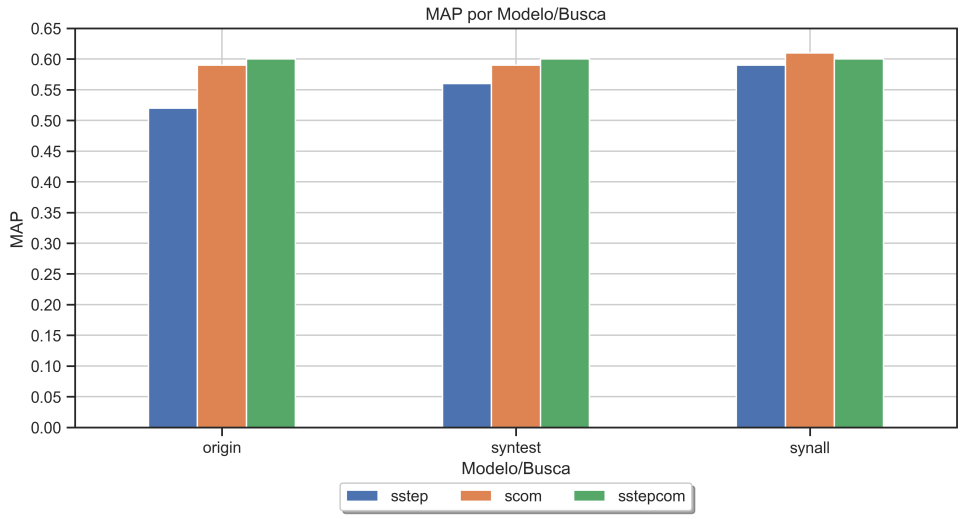

Figura 32 – MAP - Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados - Exp1

Fonte: Próprio Autor

Os resultados mostrados nas Figuras 31 e 32, mostram que, como foi visto no modelo *baseline*, a inclusão dos dados sintéticos não melhorou os resultados de forma significativa no geral, mas aumentou um pouco a pontuação de todas as buscas em média ***(+0.03)***.

Analisando as buscas de forma individual podemos ver sim uma linha de evolução de MRR e MAP. Exemplificando, vamos pegar a busca **sstep**: no modelo sem dados sintéticos **Origin** obtivemos 0.49 de MRR, incluindo os dados sintéticos somente no index , o nosso modelo **SynTest** obteve 0.52 de MRR e por fim, treinado a rede com os dados sintéticos obtivemos 0.57 de MRR, uma melhora de 8% com relação ao original.

Com isso podemos ver uma clara evolução na busca **sstep**, tendo em vista que comandos que antes não podiam ser encontrados porque não possuíam passo de teste associado agora tem um passo de teste sintético.

Também é importante salientar que, conforme citado na Subseção anterior, a inclusão de dados sintéticos não afeta a busca **scom** nos dois modelos **Origin** e **SynTest**, tendo em vista que os valores são idênticos.

Seguindo nessa comparação, também compara-se a pontuação *HitRate@n* dos três modelos, o Origin (**siamesas com dados originais**), o SynTest (**siamesas com dados sintéticos somente no index**) e o SynAll (**siamesas com dados sintéticos no index e no treino**).

A Tabela 13, mostra a descrição dos dados e a Figura 33, mostra os dados plotados em um gráfico de linhas.

MODELO	BUSCA	HR@1	HR@3	HR@5	HR@10	HR@20	HR@30	HR@40	HR@50
Origin	sstep	0.40	0.57	0.61	0.66	0.70	0.71	0.72	0.72
Origin	scom	0.44	0.65	0.72	**0.79**	**0.83**	0.84	0.86	**0.87**
Origin	sstepcom	0.44	0.65	0.72	0.78	**0.83**	**0.85**	0.86	**0.87**
SynTest	sstep	0.40	0.61	0.67	0.74	0.80	0.83	0.84	0.86
SynTest	scom	0.44	0.65	0.72	**0.79**	**0.83**	0.84	0.86	**0.87**
SynTest	sstepcom	0.43	0.65	0.72	0.78	**0.83**	**0.85**	**0.87**	**0.87**
SynAll	sstep	0.43	0.65	0.71	0.77	0.82	0.84	0.85	0.86
SynAll	scom	**0.46**	0.65	0.72	0.77	0.81	0.83	0.85	0.86
SynAll	sstepcom	0.45	**0.66**	**0.73**	0.78	0.82	0.84	0.85	0.85

Tabela 13 – Valores de HitRate@n - Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados - Exp1

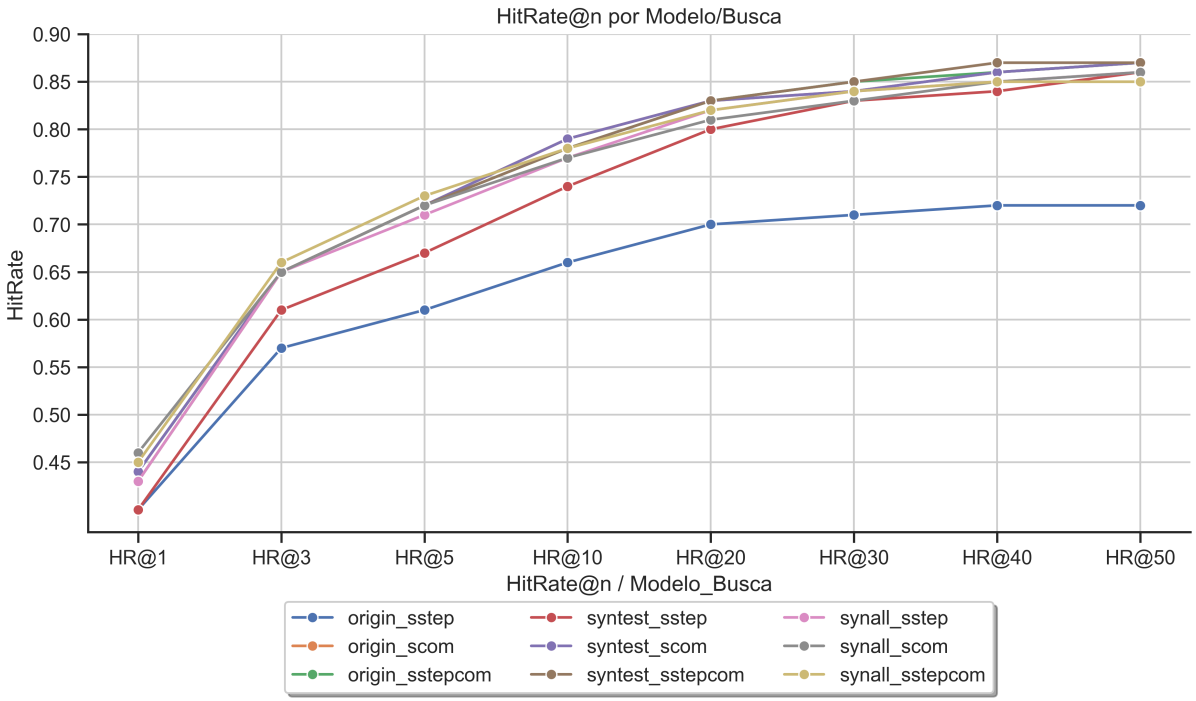

Figura 33 – HitRate@n - Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados - Exp1

Fonte: Próprio Autor

Analisando o gráfico mostrado na Figura 33 e olhando para os dados da Tabela 13, pode-se afirmar que os melhores resultados, no geral, foram com os dados sintéticos. Além disso, pode-se perceber que treinar o modelo com os dados sintéticos (**SynAll**) ajuda até o *HR@5*, conforme o valor de *n* vai subindo a partir do *n=10*, o modelo que melhor gera candidatos relevantes é o modelo com dados sintéticos somente no *index*, o **SynTest**.

5.4.4 Todos os Modelos - Exp1

Nessa etapa, compara-se todos os modelos já apresentados, tanto com e sem aumento de dados a fim de analisar qual modelo e qual busca gera o melhor *ranking* no Exp1.

A Tabela 14, mostra os dados descritivos da pontuação MRR e MAP, a Figura 34 mostra a pontuação MRR e a Figura 35, mostra a pontuação MAP de todos os nosso modelos do Exp1.

MODELO	BUSCA	MRR	MAP
BaselineSAUG	sstep	0.37	0.38
BaselineSAUG	scom	0.14	0.13
BaselineSAUG	sstepcom	0.37	0.39
BaselineCAUG	sstep	0.38	0.39
BaselineCAUG	scom	0.14	0.13
BaselineCAUG	sstepcom	0.38	0.39
Origin	sstep	0.49	0.52
Origin	scom	0.56	0.59
Origin	sstepcom	0.56	0.60
SynTest	sstep	0.52	0.56
SynTest	scom	0.56	0.59
SynTest	sstepcom	0.56	0.60
SynAll	sstep	0.55	0.59
SynAll	scom	**0.57**	**0.61**
SynAll	sstepcom	**0.57**	0.60

Tabela 14 – Valores de MRR e MAP - Todos os modelos - Exp1

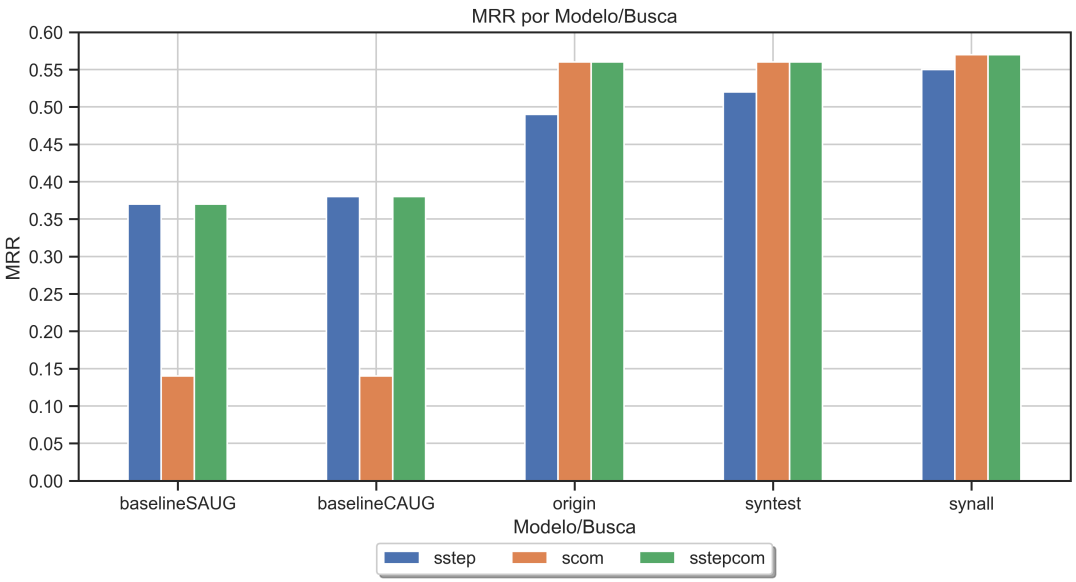

Figura 34 – MRR - Todos os modelos - Exp1

Fonte: Próprio Autor

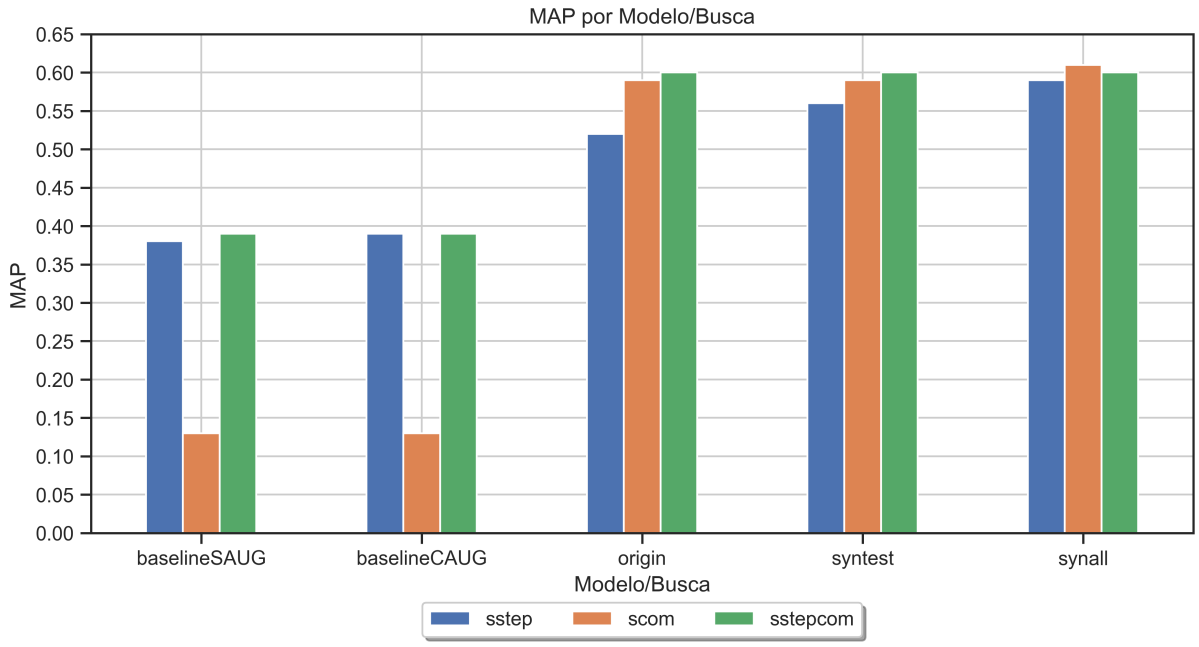

Figura 35 – MAP - Todos os modelos - Exp1

Fonte: Próprio Autor

Os resultados mostrados nas Figuras 34 e 35, mostram que apesar de não terem impacto no *baseline*, os dados sintéticos afetam positivamente os modelos baseados em redes siamesas. Comparando os modelos de redes siamesas com os modelos *baseline*, verifica-se um aumento em média de 0.19 nas buscas **sstep** e na **sstepcom**, porém o aumento mais impactante é a busca **scom** que teve um aumento de 0.43.

Pegando como exemplo o MRR da busca **scom** que é a busca que teve o maior impacto de aumento de pontuação: no *baseline*, o melhor MRR (**BaselineSAUG** e **BaselineCAUG**) é aproximadamente 0.14, isso significa que em média quando se busca um passo de teste diretamente no espaço vetorial de comandos o candidato mais relevante vem, em média, na posição 7, e no modelo de redes siamesas, o MRR é aproximadamente 0.57 no melhor cenário (**SynAll**), então esse candidato mais relevante vem, em média, entre a posição 1 e 2 do ranking.

A Tabela 15, mostra a descrição dos dados e a figura 36, mostra os dados plotados em um gráfico de linhas referente ao *HitRate@n* de todos os modelos com e sem aumento de dados.

MODELO	BUSCA	HR@1	HR@3	HR@5	HR@10	HR@20	HR@30	HR@40	HR@50
BaselineSAUG	sstep	0.27	0.44	0.48	0.53	0.57	0.60	0.62	0.63
BaselineSAUG	scom	0.08	0.16	0.19	0.26	0.32	0.37	0.41	0.40
BaselineSAUG	sstepcom	0.27	0.45	0.50	0.57	0.61	0.65	0.67	0.68
BaselineCAUG	sstep	0.28	0.45	0.49	0.57	0.63	0.66	0.69	0.71
BaselineCAUG	scom	0.08	0.16	0.19	0.26	0.32	0.37	0.41	0.44
BaselineCAUG	sstepcom	0.28	0.44	0.50	0.56	0.64	0.66	0.69	0.71
Origin	sstep	0.40	0.57	0.61	0.66	0.70	0.71	0.72	0.72
Origin	scom	0.44	0.65	0.72	**0.79**	**0.83**	0.84	0.86	**0.87**
Origin	sstepcom	0.44	0.65	0.72	0.78	**0.83**	**0.85**	0.86	**0.87**
SynTest	sstep	0.40	0.61	0.67	0.74	0.80	0.83	0.84	0.86
SynTest	scom	0.44	0.65	0.72	**0.79**	**0.83**	0.84	0.86	**0.87**
SynTest	sstepcom	0.43	0.65	0.72	0.78	**0.83**	**0.85**	**0.87**	**0.87**
SynAll	sstep	0.43	0.65	0.71	0.77	0.82	0.84	0.85	0.86
SynAll	scom	**0.46**	0.65	0.72	0.77	0.81	0.83	0.85	0.86
SynAll	sstepcom	0.45	**0.66**	**0.73**	0.78	0.82	0.84	0.85	0.85

Tabela 15 – Valores de HitRate@n - Todos os modelos - Exp1

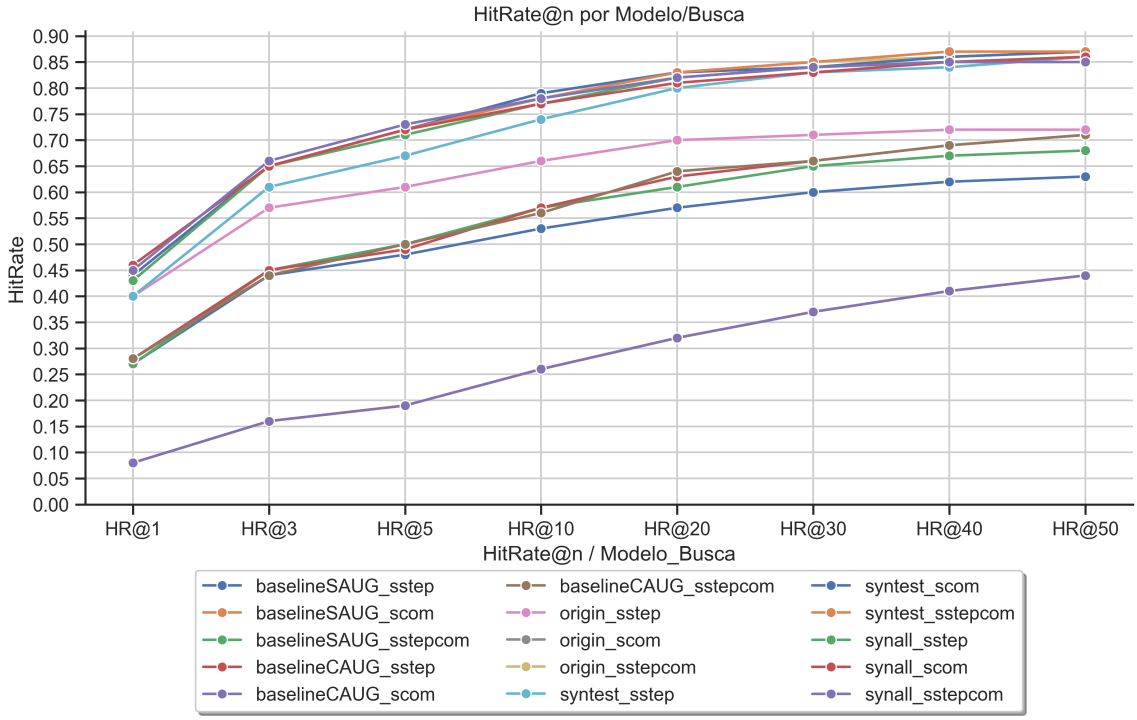

Figura 36 – HitRate@n - Todos os modelos - Exp1

Fonte: Próprio Autor

Conforme a Figura 36, e analisando os resultados, percebe-se que o modelo que gerou melhores *rankings* foram os modelos **SynTest** e **SynAll** com as buscas **sstepcom** e **sstep**, que são os modelos baseados em redes siamesas com dados sintéticos deste estudo.

Se fosse para escolher um modelo e uma busca para utilizar como o melhor, analisando

os dados, o modelo **SynAll** com a busca **sstepcom** devido à sua superioridade em termos de MRR e desempenho melhor ou competitivo em todas as outras métricas. Com os experimentos até então, podemos tirar algumas conclusões gerais.

Primeiro, os modelos de redes siamesas melhoram de forma significativa a busca dos dados de diferentes domínios, pois eles têm a capacidade de representar os dados da melhor forma e além disso, utilizando a função de perda, pode-se aproximar no espaço vetorial os pares de dados Passo de Teste e comandos de automação que tem relação.

Todos os resultados comparando *baseline* com os nossos modelos tiveram um aumento significativo nas pontuações das métricas avaliadas neste trabalho.

Outra conclusão, é de que os dados sintéticos não ajudaram de forma significativa analisando os dados gerais (em média +0.03). Porém, analisando a busca pode-se perceber que, como já era esperado, a inclusão de dados sintéticos ajudou a melhorar a busca **sstep**.

Isso deve-se ao fato de que, como esse experimento é dividido pelos passos (que foram a consulta deste trabalho), e neste estudo o *index* de alguns comandos tem passos reais e outros não, os dados sintéticos conseguiram fazer com que alguns comandos que não tinham passos de teste reais fossem encontrados.

5.4.5 Fusão de Ranking - Exp1

Nesta etapa, foram utilizados os modelos que geraram os melhores *rankings* (**SynTest** e **SynAll**) e a aplicação da fusão de *ranking* nas três buscas a fim de avaliar se a fusão impactaria nos resultados finais gerais do experimento.

A Tabela 16, mostra os dados descritivos da pontuação MRR e MAP, a Figura 37, mostra o MRR e a Figura 38, o MAP das comparações entre as buscas originais propostas e as fusões combinadas entre elas.

MODELO	BUSCA	MRR	MAP
SynTest	sstep	0.52	0.56
SynTest	scom	0.56	0.59
SynTest	sstepcom	0.56	0.60
SynTest	fusion12	0.56	0.59
SynTest	fusion13	0.55	0.59
SynTest	fusion23	0.57	**0.61**
SynTest	fusion123	0.57	**0.61**
SynAll	sstep	0.55	0.59
SynAll	scom	0.57	**0.61**
SynAll	sstepcom	0.57	0.60
SynAll	fusion12	0.57	0.60
SynAll	fusion13	0.56	0.59
SynAll	fusion23	**0.58**	**0.61**
SynAll	fusion123	0.57	**0.61**

Tabela 16 – Valores de MRR e MAP - Fusão de Ranking - Exp1

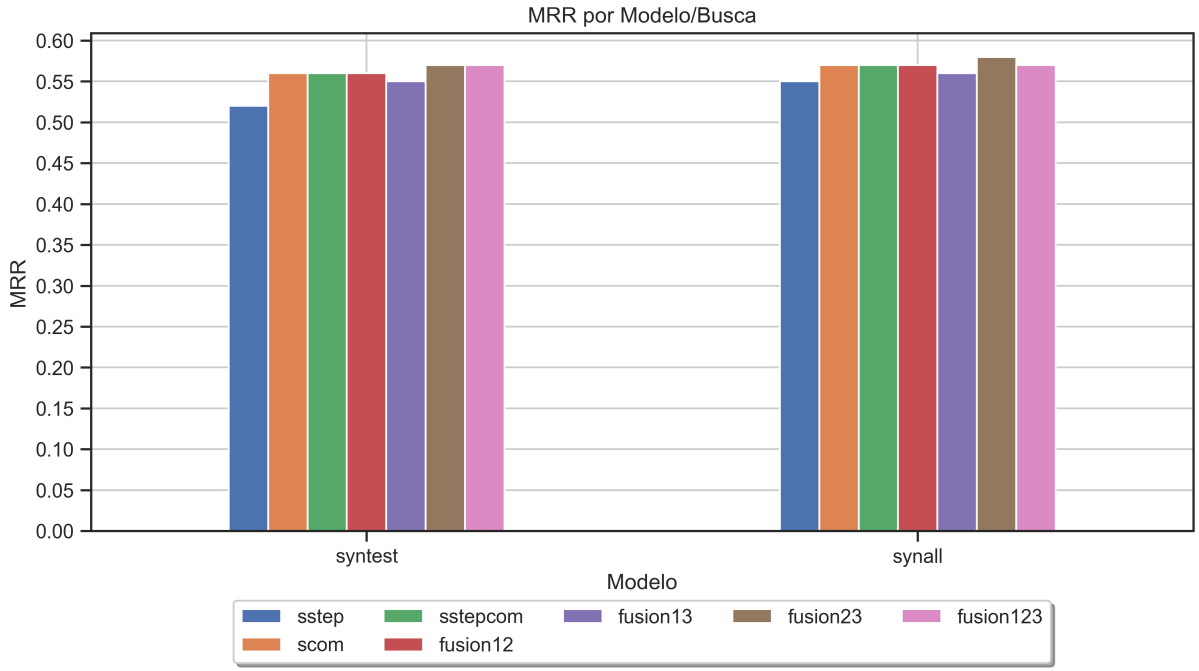

Figura 37 – MRR - Fusão de Ranking - Exp1

Fonte: Próprio Autor

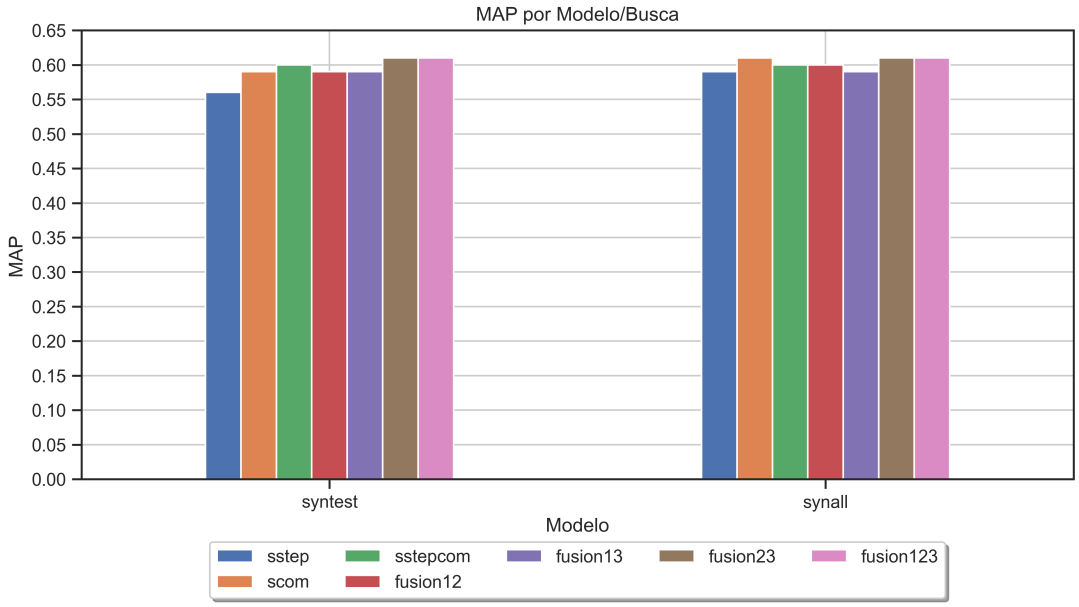

Figura 38 – MAP - Fusão de Ranking - Exp1

Fonte: Próprio Autor

Avaliando o MRR e o MAP das fusões de *ranking*, tanto na Figura 37 quanto na Figura 38, percebe-se que a fusão de *ranking* aumentou ainda mais a pontuação MRR e MAP em comparação com as buscas originais.

O melhor modelo e a melhor busca sem fusão gerou 0.57 de MRR e 0.61 de MAP (modelo **SynAll** e busca scom), já a melhor fusão (**fusion23**) aumentou +0.01 no MRR e manteve a mesma pontuação no MAP.

A Tabela 17, mostra os dados descritivos do *HitRate@n*, e a Figura 39, mostra como a fusão se saiu no *HitRate@n* em comparação com as buscas originais.

MODELO	BUSCA	HR@1	HR@3	HR@5	HR@10	HR@20	HR@30	HR@40	HR@50
SynTest	sstep	0.40	0.61	0.67	0.74	0.80	0.83	0.84	0.86
SynTest	scom	0.44	0.65	0.72	**0.79**	**0.83**	0.84	0.86	0.87
SynTest	sstepcom	0.43	0.65	0.72	0.78	**0.83**	**0.85**	**0.87**	0.87
SynTest	fusion12	0.43	0.64	0.72	0.78	0.82	0.84	0.86	0.87
SynTest	fusion13	0.43	0.64	0.70	0.77	0.81	**0.85**	**0.87**	**0.88**
SynTest	fusion23	0.46	0.66	0.71	0.78	0.82	**0.85**	0.86	0.87
SynTest	fusion123	0.44	0.66	0.71	0.78	**0.83**	**0.85**	0.86	**0.88**
SynAll	sstep	0.43	0.65	0.71	0.77	0.82	0.84	0.85	0.86
SynAll	scom	0.46	0.65	0.72	0.77	0.81	0.83	0.85	0.86
SynAll	sstepcom	0.45	0.66	**0.73**	0.78	0.82	0.84	0.85	0.85
SynAll	fusion12	0.46	0.65	0.71	0.77	0.81	0.84	0.85	0.86
SynAll	fusion13	0.45	0.65	0.71	0.77	0.80	0.81	0.83	0.84
SynAll	fusion23	**0.47**	**0.67**	0.72	0.78	0.81	0.83	0.84	0.85
SynAll	fusion123	0.46	0.66	0.71	0.77	0.81	0.83	0.84	0.85

Tabela 17 – Valores de HitRate@n - Fusão de Ranking - Exp1

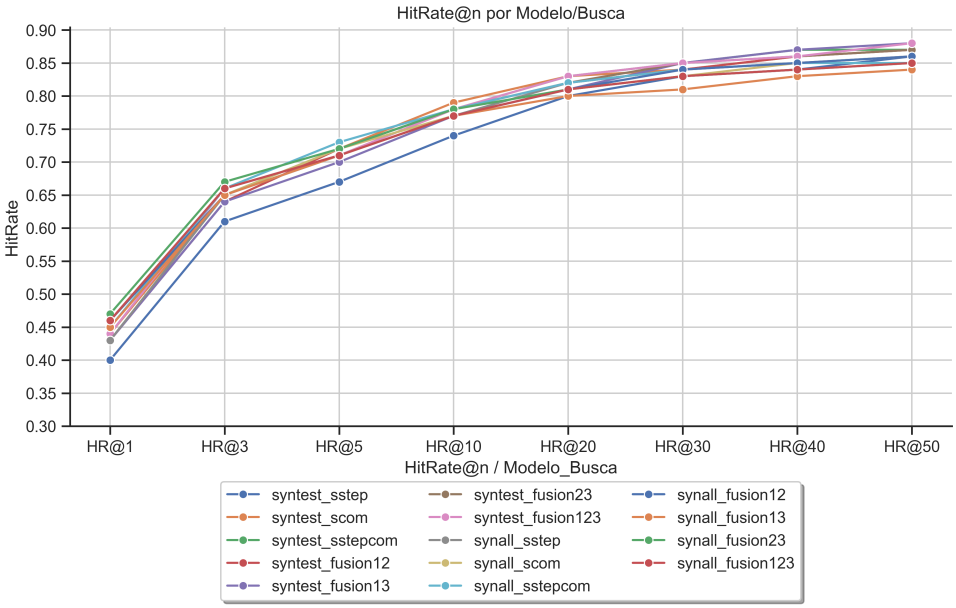

Figura 39 – HitRate@n - Fusão de Ranking - Exp1

Fonte: Próprio Autor

Analisando o *HitRate* da Figura 39, pode-se afirmar que a fusão dos *ranking* gerou ganho em comparação às buscas individuais. Pode-se perceber que a fusão é melhor no n 1 e 3, nos demais valores de n as buscas **scom** e **sstepcom** ganham e algumas vezes igualam a pontuação gerada pela fusão.

Em suma, o modelo **SynAll** recupera mais comandos relevantes nas posições mais altas (posição 5 ou menor) e o modelo **SynTest** consegue recuperar mais comandos relevantes nas 50 primeiras posições.

5.5 Resultados - Exp2

5.5.1 Dados Originais - Exp2

Neste segundo experimento, como anteriormente explicado, tem como intenção, avaliar o efeito dos métodos propostos em cenários onde o comando correto ainda não foi automatizado, e, por conseguinte, não possui nenhum passo de teste relacionado para auxiliar na sua recuperação.

Um conjunto de comandos de automação é separado como conjunto de teste, e seus passos de teste relacionados são removidos da base de dados e utilizados apenas como *strings* de busca. Desta forma, os comandos corretos para cada busca serão sempre comandos sem passo de teste associado.

Inicialmente, compara-se os resultados do *baseline* com o modelo proposto baseado em redes siamesas utilizando os dados originais (sem dados sintéticos). A ideia de mostrar essa comparação é validar se este modelo proposto baseado em redes siamesas realmente ajuda na representação de passos de teste e comandos de automação, aproximando os pares no mesmo espaço vetorial de busca.

A Tabela 18, mostra os dados descritivos da pontuação MRR e MAP, a Figura 40, mostra a pontuação MRR e a Figura 41, mostra a pontuação MAP dessa comparação.

MODELO	BUSCA	MRR	MAP
BaselineSAUG	sstep	0.00	0.00
BaselineSAUG	scom	0.11	0.11
BaselineSAUG	sstepcom	0.04	0.04
Origin	sstep	0.00	0.00
Origin	scom	**0.28**	**0.28**
Origin	sstepcom	0.22	0.22

Tabela 18 – Valores de MRR e MAP dos modelos com dados originais - Exp2

Figura 40 – MRR dos modelos com dados originais - Exp2

Fonte: Próprio Autor

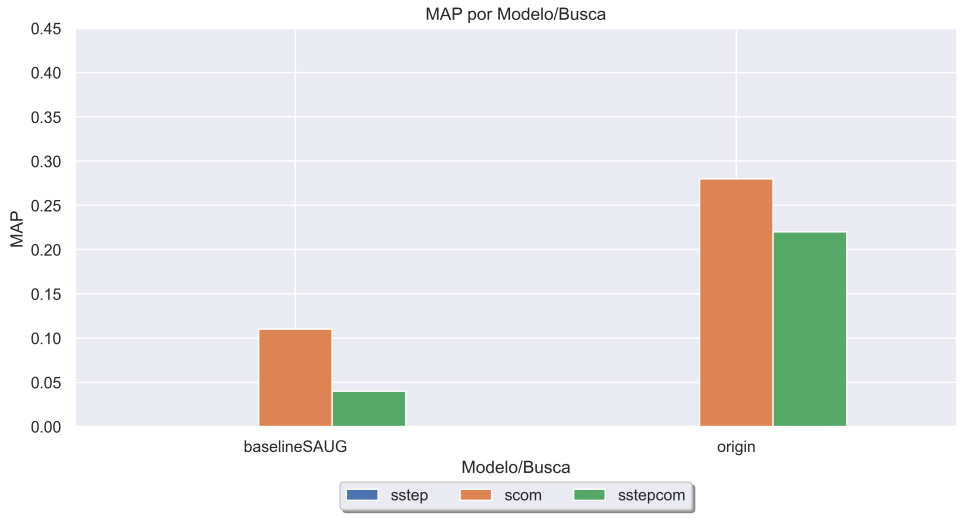

Figura 41 – MAP dos modelos com dados originais - Exp2

Fonte: Próprio Autor

Os resultados mostrados nas Figuras 40 e 41, mostram que a utilização do modelo proposto baseado em redes siamesas, como foi concluído com o Exp1, realmente ajuda na representação dos passos de teste e dos comandos no mesmo espaço e melhora as pontuações de forma significativa tanto no MRR quanto no MAP.

Além disso, analisando a busca **sstep**, pode-se perceber que as pontuações ficaram zeradas tanto no *baseline* quanto no Origin. Isso se deve ao fato de que os comandos corretos

para cada busca não possuem um passo de teste associado, sendo impossível recuperá-los neste cenário de busca por passo de teste.

Também compara-se a pontuação *HitRate@n* dos modelos *baseline* e Origin.

A Tabela 19, mostra a descrição dos dados e a Figura 42, mostra os dados plotados em um gráfico de linhas para verificar a evolução de cada busca a medida que se aumenta o *top-n* candidatos.

MODELO	BUSCA	HR@1	HR@3	HR@5	HR@10	HR@20	HR@30	HR@40	HR@50
BaselineSAUG	sstep	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
BaselineSAUG	scom	0.06	0.13	0.17	0.23	0.29	0.33	0.35	0.37
BaselineSAUG	sstepcom	0.01	0.04	0.07	0.11	0.17	0.21	0.24	0.26
Origin	sstep	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Origin	scom	**0.14**	**0.34**	**0.45**	**0.56**	**0.64**	**0.68**	**0.70**	**0.72**
Origin	sstepcom	0.09	0.27	0.40	0.52	0.60	0.64	0.67	0.67

Tabela 19 – Valores de HitRate@n dos modelos com dados originais - Exp2

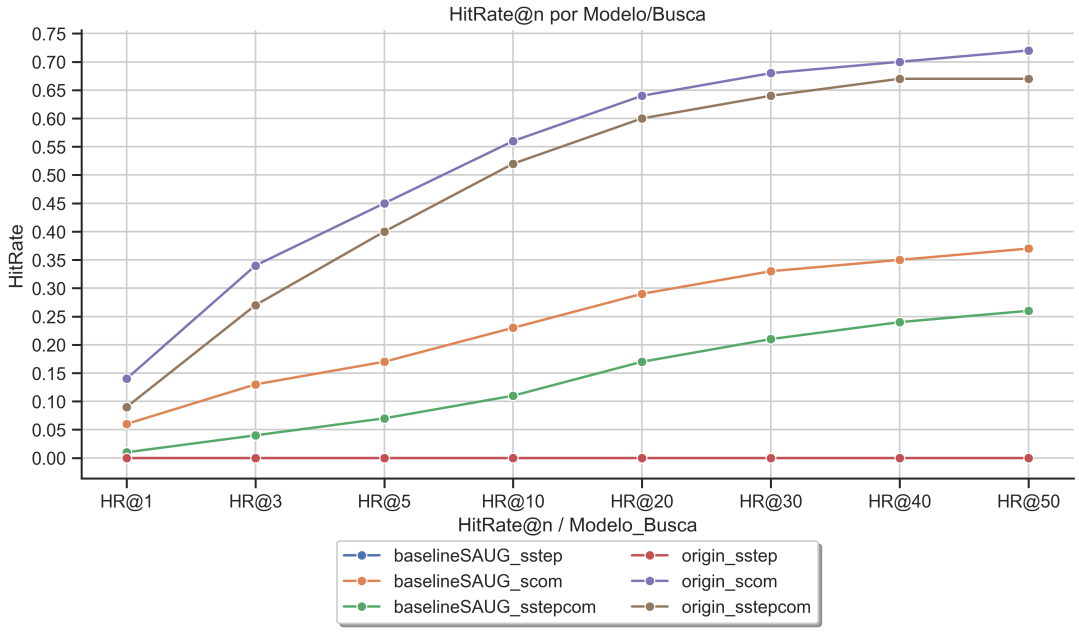

Figura 42 – HitRate@n dos modelos com dados originais - Exp2

Fonte: Próprio Autor

Analisando o gráfico mostrado na Figura 42, pode-se ver que o modelo **Origin** com a busca **scom** foi o que gerou o melhor *ranking*, e se manteve como melhores resultados, mesmo aumentando o valor de *top-n*, validando ainda mais que o modelo siamesa é a melhor abordagem a seguir neste trabalho.

Para analisar o porquê da busca **scom** ser a melhor nesse caso, pode-se compará-la com as outras buscas. A busca **sstep** teve uma pontuação zerada porque não temos pares reais do comando no espaço de busca, e a busca **sstepcom** também teve seu desempenho um pouco abaixo da busca **scom**, já que ela concatena as duas buscas, e como a busca **sstep** zerou a pontuação, ela atrapalhou alguns comandos a serem buscados no *ranking* final.

5.5.2 Dados Originais + Sintéticos - Exp2

Nesta Seção, compara-se o impacto que os dados sintéticos tiveram na tarefa de busca. Para uma melhor organização, primeiro foi analisado o impacto dos dados sintéticos no *baseline* e na sequência da rede siamesa proposta.

5.5.2.1 Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados

Inicialmente, avaliou-se o impacto das técnicas de aumento de dados sobre o modelo *baseline*. Com isso, a intenção era validar se a etapa de geração de dados sintéticos ajudou a buscar os comandos que não tinham um par de Passo de Teste associado, que no caso do Exp2, eram a totalidade dos comandos corretos.

A Tabela 20, mostra os dados descritivos da pontuação MRR e MAP, a Figura 43, mostra a pontuação MRR e a Figura 44, mostra a pontuação MAP dessa comparação.

MODELO	BUSCA	MRR	MAP
BaselineSAUG	sstep	0.00	0.00
BaselineSAUG	scom	**0.11**	**0.11**
BaselineSAUG	sstepcom	0.04	0.04
BaselineCAUG	sstep	0.03	0.03
BaselineCAUG	scom	**0.11**	**0.11**
BaselineCAUG	sstepcom	0.06	0.06

Tabela 20 – Valores de MRR e MAP - Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados - Exp2

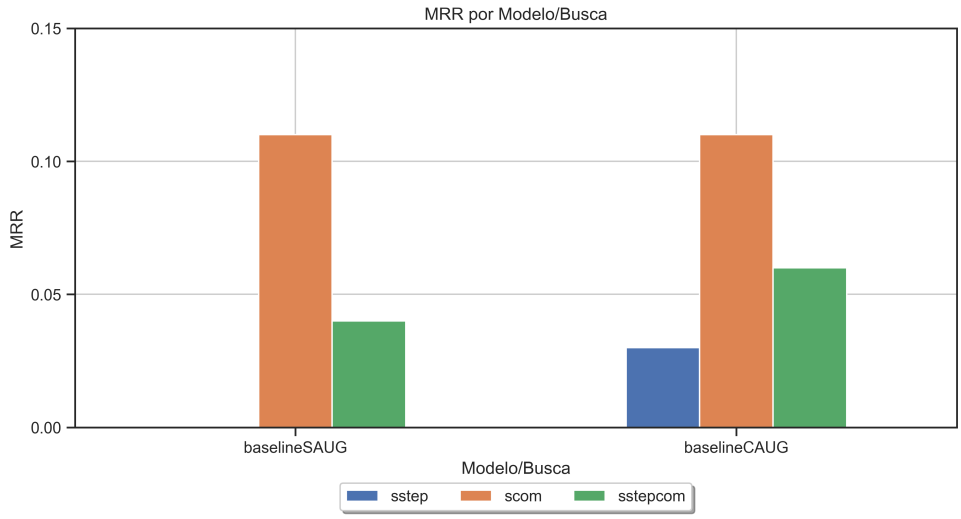

Figura 43 – MRR - Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados - Exp2

Fonte: Próprio Autor

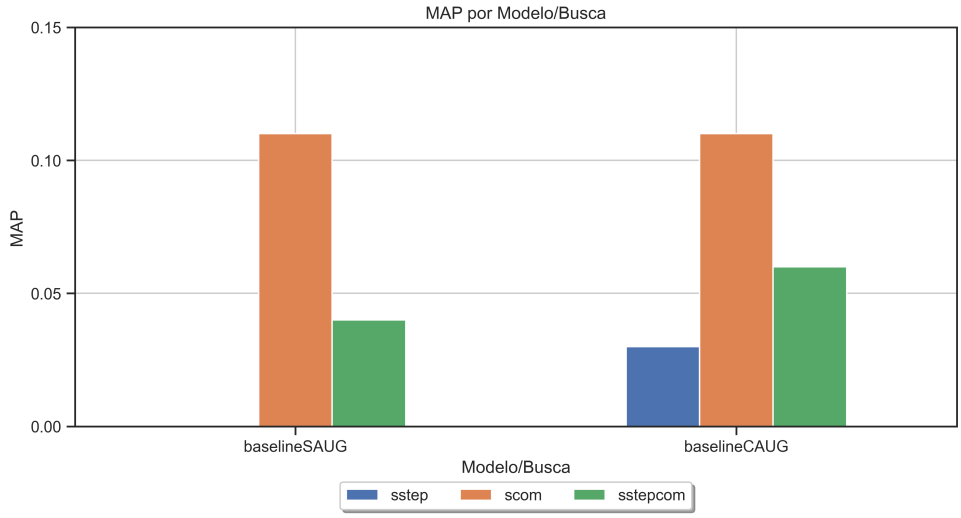

Figura 44 – MAP - Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados - Exp2

Fonte: Próprio Autor

Os resultados mostrados nas Figuras 43 e 44, mostram que a inclusão dos dados sintéticos na busca no nosso *baseline* ajudou a busca do **sstep** e do **sstepcom**, tendo em vista que essas duas buscas envolvem o Passo de Teste e, como já citado, os comandos a serem encontrados no espaço de busca não contém os passos reais associados.

Com isso, pode-se concluir que, nessa análise do *baseline*, que os dados sintéticos ajudaram a buscar os comandos via adição de passo relacionados.

Seguindo nessa comparação, também comparou-se a pontuação *HitRate@n* dos dois modelos, o *baseline* sem aumento de dados e o *baseline* com aumento de dados. A Tabela 21, mostra a descrição dos dados e a Figura 45, mostra os dados plotados em um gráfico de linhas para verificar a evolução de cada busca à medida que se aumenta o *top-n* candidatos.

MODELO	BUSCA	HR@1	HR@3	HR@5	HR@10	HR@20	HR@30	HR@40	HR@50
BaselineSAUG	sstep	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
BaselineSAUG	scom	**0.06**	**0.13**	**0.17**	**0.23**	**0.29**	**0.33**	**0.35**	**0.37**
BaselineSAUG	sstepcom	0.01	0.04	0.07	0.11	0.17	0.21	0.24	0.26
BaselineCAUG	sstep	0.01	0.03	0.05	0.09	0.14	0.18	0.21	0.23
BaselineCAUG	scom	**0.06**	**0.13**	**0.17**	**0.23**	**0.29**	**0.33**	**0.35**	**0.37**
BaselineCAUG	sstepcom	0.02	0.06	0.09	0.15	0.22	0.26	0.29	0.32

Tabela 21 – Valores de HitRate@n - Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados - Exp2

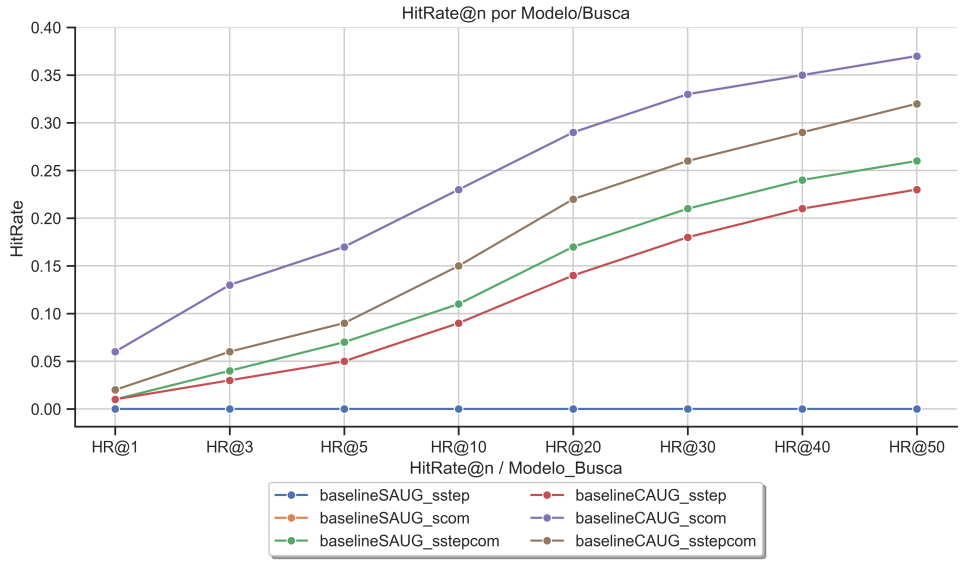

Figura 45 – HitRate@n - Baseline Sem Aumento de Dados x Baseline Com Aumento de Dados - Exp2

Fonte: Próprio Autor

Analisando o gráfico mostrado na Figura 45, pode-se afirmar que, conforme conclusões do MRR e do MAP, os dados sintéticos de modo geral, ajudaram a busca **sstep** e **sstepcom**.

Os melhores resultados gerais foram na busca por **scom**, independente do aumento de dados. Isso deve-se ao fato de que, como os comandos do treino não tem um passo de teste real associado, a busca tende a buscar diretamente pelos comandos já que os mesmos estão representados no espaço de busca.

5.5.2.2 Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados

Nesta etapa, compara-se o impacto que os dados sintéticos tiveram na tarefa de busca do modelo baseado em redes siamesas no Exp2. A Tabela 22, mostra os dados descritivos da pontuação MRR e MAP, a Figura 46, mostra a pontuação MRR e a Figura 47, mostra a pontuação MAP dessa comparação.

MODELO	BUSCA	MRR	MAP
Origin	sstep	0.00	0.00
Origin	scom	**0.28**	**0.28**
Origin	sstepcom	0.22	0.22
SynTest	sstep	0.15	0.15
SynTest	scom	**0.28**	**0.28**
SynTest	sstepcom	0.23	0.24
SynAll	sstep	0.16	0.16
SynAll	scom	0.24	0.24
SynAll	sstepcom	0.19	0.19

Tabela 22 – Valores de MRR e MAP - Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados - Exp2

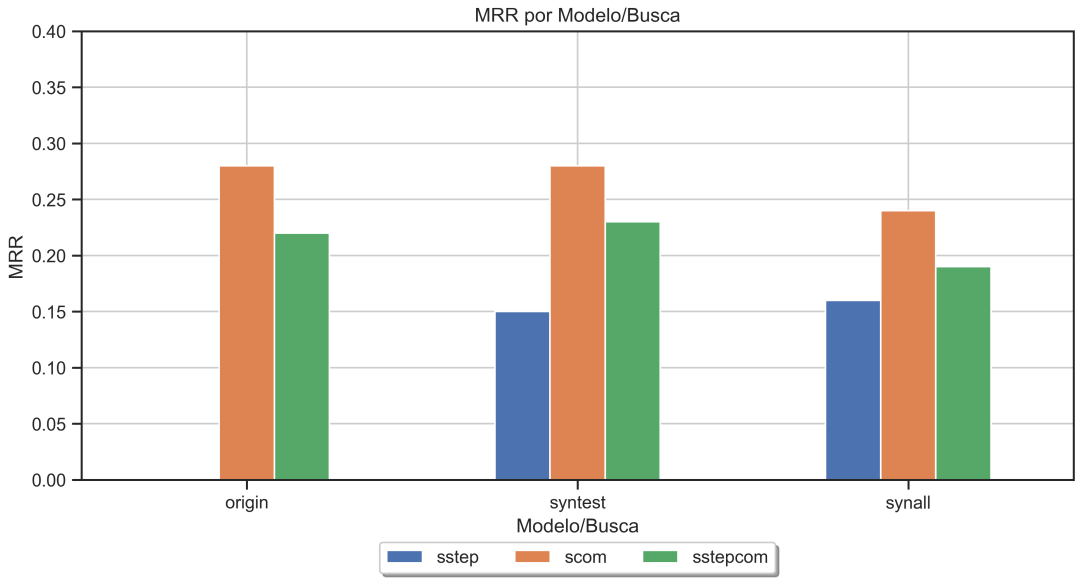

Figura 46 – MRR - Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados - Exp2

Fonte: Próprio Autor

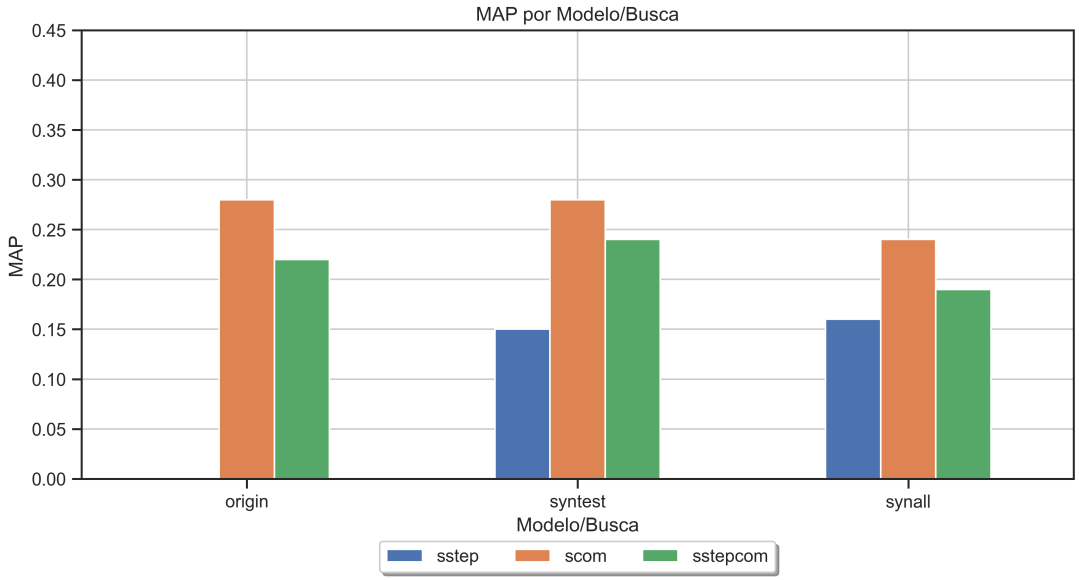

Figura 47 – MAP - Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados - Exp2

Fonte: Próprio Autor

Os resultados mostrados nas Figuras 46 e 47, mostram que os dados sintéticos melhoram a busca **sstep** tendo em vista que, como visto na comparação do *baseline*, os comandos que antes não podiam ser encontrados por não ter um par de passo de teste associado agora tem um passo de teste sintético.

Pôde-se ver também que apesar de bem pouco, houve um aumento na pontuação MRR na busca **sstepcom** do modelo Origin para o modelo SynTest.

Contudo, ao se comparar com o modelo onde os dados sintéticos são usados também no treinamento, SynAll, este último não conseguiu alcançar resultados tão bons quanto a versão sem treinamento com aumento de dados, SynTest.

Pôde-se então concluir que, os dados sintéticos nesse caso ajudaram na busca somente quando os mesmos são apenas representados no índices, e que os mesmos ao serem utilizados no treinamento adicionaram ruído que não levou a ganhos na representação geral dos pares passo de teste-comando.

Seguindo nessa comparação, também comparou-se a pontuação *HitRate@n* dos três modelos, o Origin (**siamesas com dados originais**), o SynTest (**siamesas com dados sintéticos somente no índice**) e o SynAll (**siamesas com dados sintéticos no índice e no treino**).

A Tabela 23, mostra a descrição dos dados e a Figura 48, mostra os dados plotados em um gráfico de linhas. Também é importante salientar que, como os resultados da busca **scom**

nos modelos Origin e SynTest são idênticos, no gráfico da Figura 48 eles se sobrepõem.

MODELO	BUSCA	HR@1	HR@3	HR@5	HR@10	HR@20	HR@30	HR@40	HR@50
Origin	sstep	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Origin	scom	**0.14**	**0.34**	**0.45**	**0.56**	**0.64**	**0.68**	**0.70**	**0.72**
Origin	sstepcom	0.09	0.27	0.40	0.52	0.60	0.64	0.67	0.67
SynTest	sstep	0.04	0.19	0.27	0.38	0.49	0.54	0.57	0.59
SynTest	scom	**0.14**	**0.34**	**0.45**	**0.56**	**0.64**	**0.68**	**0.70**	**0.72**
SynTest	sstepcom	0.09	0.30	0.42	0.53	0.62	0.67	**0.70**	**0.72**
SynAll	sstep	0.07	0.18	0.26	0.38	0.46	0.49	0.51	0.52
SynAll	scom	0.13	0.29	0.38	0.46	0.54	0.59	0.62	0.64
SynAll	sstepcom	0.08	0.23	0.33	0.43	0.51	0.54	0.55	0.55

Tabela 23 – Valores de HitRate@n - Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados - Exp2

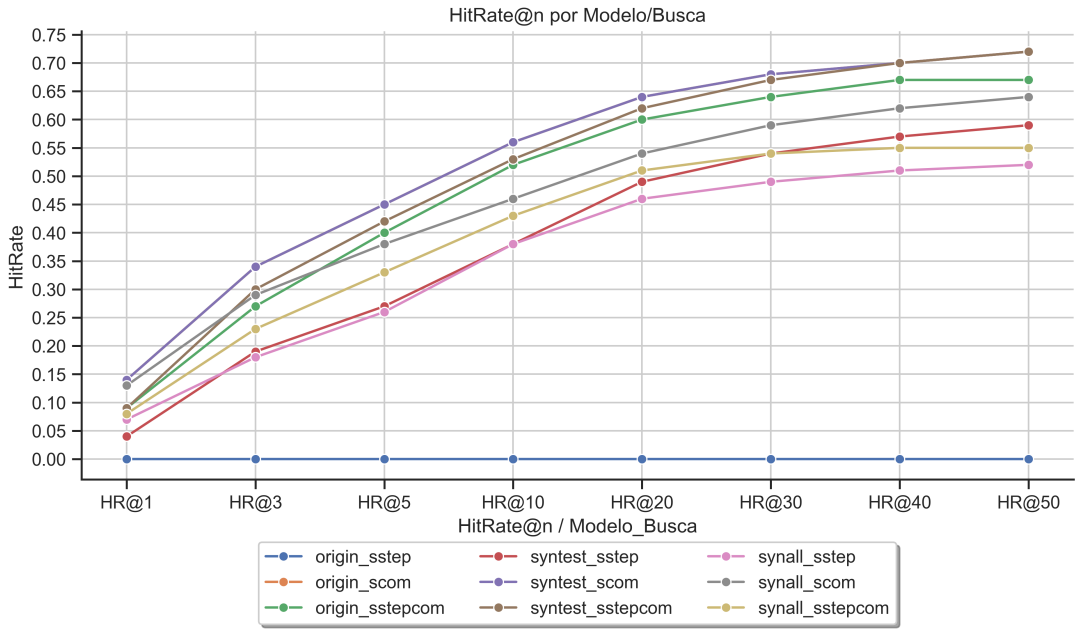

Figura 48 – HitRate@n - Siamesas Sem Aumento de Dados x Siamesas Com Aumento de Dados - Exp2

Fonte: Próprio Autor

Analisando o gráfico mostrado na Figura 48, pode-se afirmar que os melhores resultados foram com a busca **scom**, validando a proposta do trabalho: buscar passos de teste diretamente com o comando de automação associado ao mesmo.

Outra observação é em relação ao modelo **SynAll**, que em comparação aos outros modelos teve as pontuações baixas validando o que havíamos concluído na subsecção anterior: que os dados sintéticos, quando usados no treino do modelo, atrapalham a representação e pioram os resultados da busca.

5.5.3 Todos os Modelos - Exp2

Nessa etapa, comparou-se todos os modelos já apresentados no Exp2, tanto com e sem aumento de dados, a fim de analisar qual modelo e qual busca gera o melhor *ranking* no Exp2, compilando todas as conclusões tiradas das Subseções anteriores. A Tabela 24, mostra os dados descritivos da pontuação MRR e MAP, a Figura 49, mostra a pontuação MRR e a Figura 50, mostra a pontuação MAP dessa comparação.

MODELO	BUSCA	MRR	MAP
BaselineSAUG	sstep	0.00	0.00
BaselineSAUG	scom	0.11	0.11
BaselineSAUG	sstepcom	0.04	0.04
BaselineCAUG	sstep	0.03	0.03
BaselineCAUG	scom	0.11	0.11
BaselineCAUG	sstepcom	0.06	0.06
Origin	sstep	0.00	0.00
Origin	scom	**0.28**	**0.28**
Origin	sstepcom	0.22	0.22
SynTest	sstep	0.15	0.15
SynTest	scom	**0.28**	**0.28**
SynTest	sstepcom	0.23	0.24
SynAll	sstep	0.16	0.16
SynAll	scom	0.24	0.24
SynAll	sstepcom	0.19	0.19

Tabela 24 – Valores de MRR e MAP - Todos os modelos - Exp2

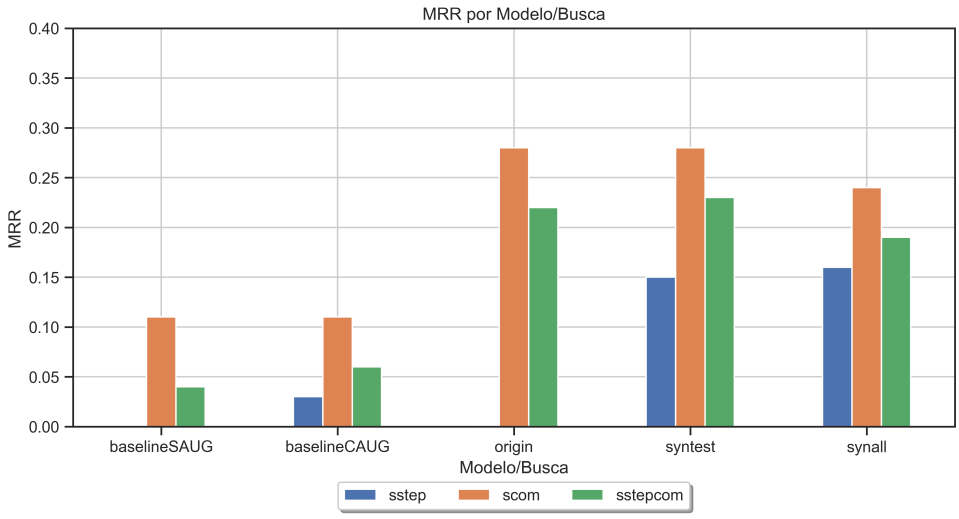

Figura 49 – MRR - Todos os modelos - Exp2

Fonte: Próprio Autor

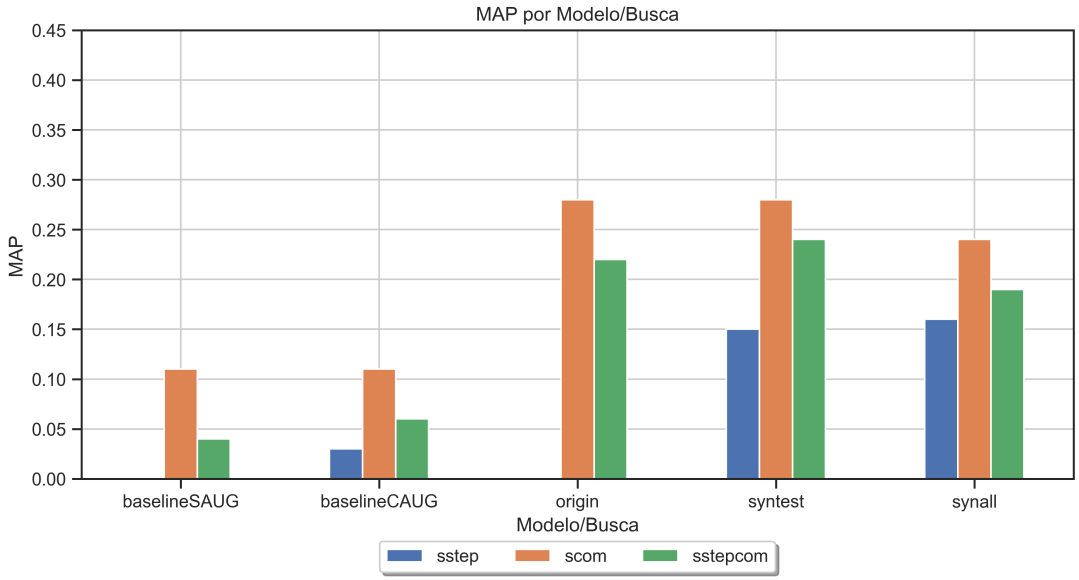

Figura 50 – MAP - Todos os modelos - Exp2

Fonte: Próprio Autor

Conforme as Figuras 49 e 50, pode-se tirar algumas conclusões baseado nas pontuações MRR e MAP. No Exp2, os dados sintéticos tiveram um grande impacto em comparação aos modelos sem aumento de dados na busca **sstep**. Isso está ligado a falta de representação de pares de passos de teste associados aos comandos de automação que estão no *Index*.

Também pode-se afirmar que, os dados sintéticos nesse experimento só ajudaram na busca **scom** e **sstepcom** quando os mesmos não foram usados para treinamento, mas sim somente para indexação. Pode-se dizer que a razão disso é que os dados sintéticos, quando incluídos na base de treinamento, está atrapalhando na qualidade da representação dos comandos no espaço de *embeddings*.

E por fim, conclui-se no Exp1, os modelos baseados em redes siamesas foram os que obtiveram os melhores *rankings* em todas as buscas comparando com o *baseline*.

A Tabela 25, mostra a descrição dos dados e a Figura 51, mostra os dados plotados em um gráfico de linhas referente ao *HitRate@n* de todos os modelos com e sem aumento de dados.

MODELO	BUSCA	HR@1	HR@3	HR@5	HR@10	HR@20	HR@30	HR@40	HR@50
BaselineSAUG	sstep	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
BaselineSAUG	scom	0.06	0.13	0.17	0.23	0.29	0.33	0.35	0.37
BaselineSAUG	sstepcom	0.01	0.04	0.07	0.11	0.17	0.21	0.24	0.26
BaselineCAUG	sstep	0.01	0.03	0.05	0.09	0.14	0.18	0.21	0.23
BaselineCAUG	scom	0.06	0.13	0.17	0.23	0.29	0.33	0.35	0.37
BaselineCAUG	sstepcom	0.02	0.06	0.09	0.15	0.22	0.26	0.29	0.32
Origin	sstep	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Origin	scom	**0.14**	**0.34**	**0.45**	**0.56**	**0.64**	**0.68**	**0.70**	**0.72**
Origin	sstepcom	0.09	0.27	0.40	0.52	0.60	0.64	0.67	0.67
SynTest	sstep	0.04	0.19	0.27	0.38	0.49	0.54	0.57	0.59
SynTest	scom	**0.14**	**0.34**	**0.45**	**0.56**	**0.64**	**0.68**	**0.70**	**0.72**
SynTest	sstepcom	0.09	0.30	0.42	0.53	0.62	0.67	**0.70**	**0.72**
SynAll	sstep	0.07	0.18	0.26	0.38	0.46	0.49	0.51	0.52
SynAll	scom	0.13	0.29	0.38	0.46	0.54	0.59	0.62	0.64
SynAll	sstepcom	0.08	0.23	0.33	0.43	0.51	0.54	0.55	0.55

Tabela 25 – Valores de HitRate@n - Todos os modelos - Exp2

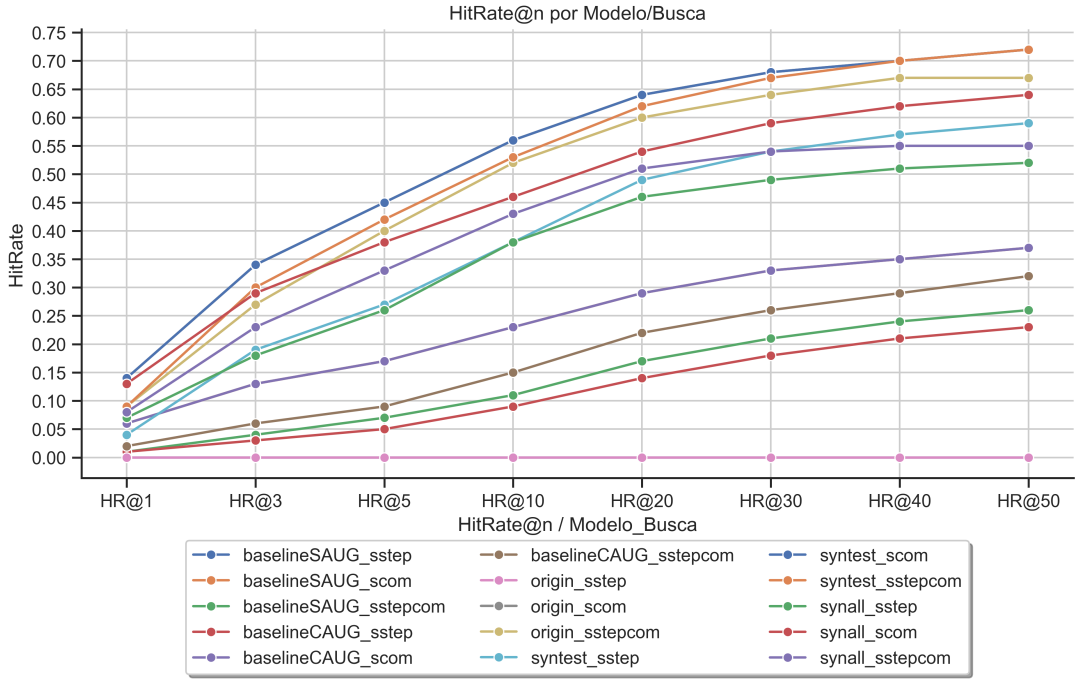

Figura 51 – HitRate@n - Todos os modelos - Exp2

Fonte: Próprio Autor

Conforme vimos na duas Figura acima 51 e analisando as conclusões tiradas da pontuação MRR e MAP, pode-se afirmar que o melhor modelo foi o **SynTest** e o **Origin** combinado com a busca **scom** que, mesmo como valor de n aumentando no *HitRate* ele se manteve com as melhores pontuações e de forma disparada.

5.5.4 Fusão de Ranking - Exp2

Nesta etapa, foram utilizados os modelos que geraram os melhores *ranking* (**SynTest** e **Origin**) e aplicado a fusão de *ranking* nas três buscas a fim de avaliar se a fusão impactaria nos resultados finais gerais do experimento.

Nessa fusão, foram desconsiderados a fusão que continha o *rank* (1) **sstep**, pois já que a divisão foi feita por comandos nenhum passo de teste potencialmente correto foi recuperado e o *ranking* **sstep** zerou. Assim, a fusão usava esse *ranking* como uma lista negativa, porém esse comportamento não condiz com o que temos em produção já que podemos ter os passos corretos nessa busca.

A Tabela 26, mostra os dados descritivos da pontuação MRR e MAP, a Figura 52, mostra o MRR e a Figura 53, o MAP das comparações entre as buscas originais propostas e as fusões combinadas entre elas.

MODELO	BUSCA	MRR	MAP
Origin	sstep	0.00	0.00
Origin	scom	0.28	0.28
Origin	sstepcom	0.22	0.22
Origin	fusion23	**0.31**	**0.32**
SynTest	sstep	0.15	0.15
SynTest	scom	0.28	0.28
SynTest	sstepcom	0.23	0.24
SynTest	fusion12	0.28	0.29
SynTest	fusion13	0.28	0.28
SynTest	fusion23	0.28	0.28
SynTest	fusion123	**0.31**	0.31

Tabela 26 – Valores de MRR e MAP - Fusão de Ranking - Exp2

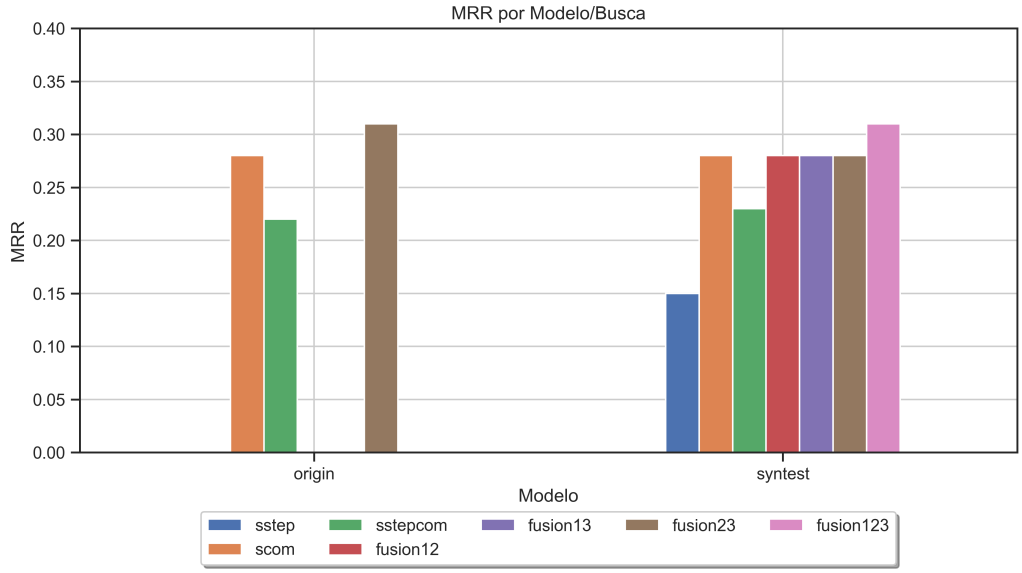

Figura 52 – MRR - Fusão de Ranking - Exp2

Fonte: Próprio Autor

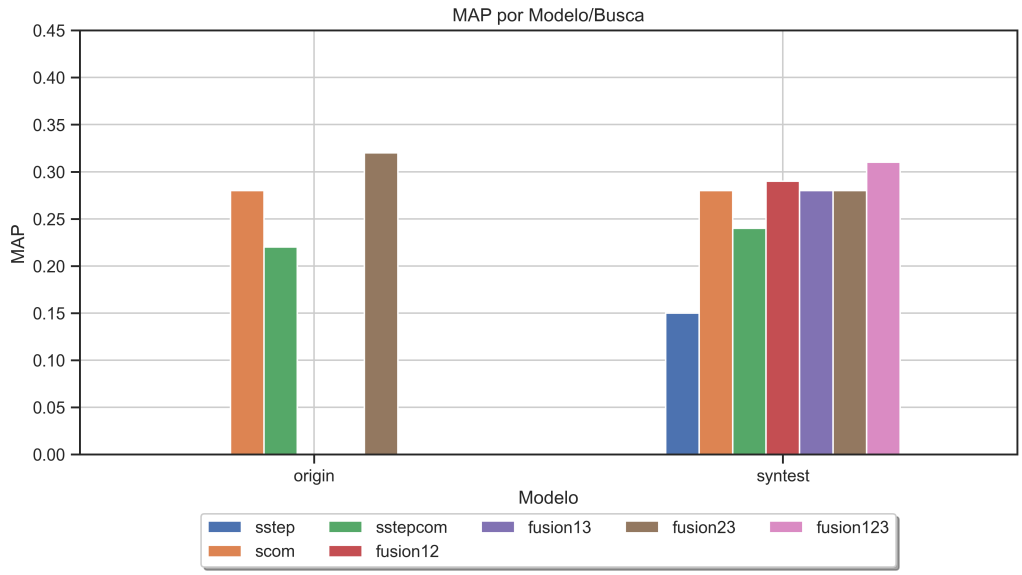

Figura 53 – MAP - Fusão de Ranking - Exp2

Fonte: Próprio Autor

Avaliando o MRR e o MAP das fusões de *ranking*, tanto na Figura 52 quanto na Figura 53, podemos perceber que que a fusão de *ranking* melhorou as pontuações, sendo que a única fusão avaliada no modelo Origin foi a que gerou os melhores resultados, e no modelo SynTest a fusão combinada das três buscas foi a que se saiu melhor.

Como já citado, as outras três fusões do Origin não foram avaliadas tendo em vista que elas continham a busca **sstep**, porém no *model* SynTest não havia impedimento já que a busca **sstep** gerou resultados.

A Tabela 27 e a Figura 54, mostram como a fusão se saiu no *HitRate@n* em comparação com as buscas originais.

MODELO	BUSCA	HR@1	HR@3	HR@5	HR@10	HR@20	HR@30	HR@40	HR@50
Origin	sstep	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Origin	scom	0.14	0.34	0.45	**0.56**	0.64	**0.68**	0.70	0.72
Origin	sstepcom	0.09	0.27	0.40	0.52	0.60	0.64	0.67	0.67
Origin	fusion23	0.19	**0.37**	**0.47**	0.55	0.63	0.67	0.70	0.72
SynTest	sstep	0.04	0.19	0.27	0.38	0.49	0.54	0.57	0.59
SynTest	scom	0.14	0.34	0.45	**0.56**	0.64	**0.68**	0.70	0.72
SynTest	sstepcom	0.09	0.30	0.42	0.53	0.62	0.67	0.70	0.72
SynTest	fusion12	0.17	0.32	0.41	0.53	0.63	0.67	0.71	0.72
SynTest	fusion13	0.17	0.32	0.39	0.51	0.63	0.67	0.70	0.71
SynTest	fusion23	0.15	0.33	0.44	0.55	**0.65**	**0.68**	**0.72**	0.74
SynTest	fusion123	**0.20**	0.35	0.42	0.54	0.63	**0.68**	0.71	**0.75**

Tabela 27 – Valores de HitRate@n - Fusão de Ranking - Exp2

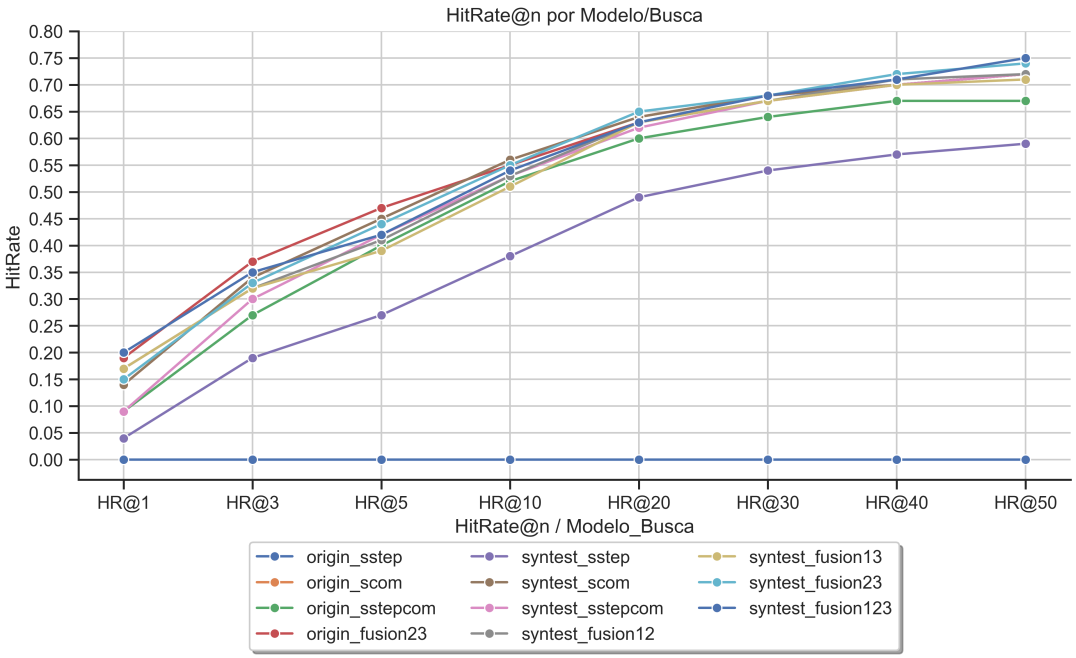

Figura 54 – HitRate@n - Fusão de Ranking - Exp2

Fonte: Próprio Autor

Novamente, como foi visto no MRR e no MAP, foi obtido sim uma melhora grande com a fusão de *rankings*, principalmente nos *rankings* que continham a busca **scom** que é o *ranking* 2.

Além disso, a busca **scom** obteve bons resultados sendo que os seus resultados ficaram próximos aos resultados obtidos com a fusão de *rankings*.

No fim das contas, os melhores resultados foram com a fusão de *rankings*, com o modelo **SynTest** gerando os melhores resultados, e a busca **scom** gerando resultados parecidos.

5.6 Considerações Finais do Capítulo 5 - Experimentos

As conclusões tiradas dos dois experimentos variaram um pouco, mas a conclusão unânime é que utilizar Redes Siamesas para representar tanto passos de teste quanto comandos no mesmo espaço vetorial ajuda, de forma significativa, a gerar *rankings* melhores no que diz respeito à busca de comandos de automação dado um passo de teste de entrada.

Nos dois experimentos, as redes siamesas superaram o *baseline* e mais do que isso, ajudaram a buscar os comandos diretamente pelos seus passos, que era o nosso objetivo inicial.

Como em produção não temos todos os comandos automatizados, ou seja, nem todos os comandos tem um passo de teste associado foi necessário criar dados sintéticos para que fosse possível encontrar esses comandos também pela busca sstep, já que esses comandos sem passo só poderiam ser encontrados pela busca scom. E nos dois experimentos, a inclusão de dados sintéticos ajudou a busca sstep a gerar melhores *rankings* tendo em vista que, principalmente no Exp2 comandos que antes não poderiam ser buscados agora podem.

No Exp1 por exemplo, que alguns comandos tem passo associado houve melhora de $\sim$0.07 no MRR, porém no Exp2 que a totalidade dos comandos não tinha representação a melhora foi de $\sim$0.16 no MRR.

Tendo todos os resultados consolidados, aplicou-se a fusão de *ranking* a fim de validar se poderíamos melhorar a ordenação de respostas, unindo nossas três buscas.

Nos dois experimentos a fusão de *ranking* obteve os melhores resultados, aumentando em $\sim$0.03 a pontuação MRR. No Exp1 e no Exp2 a união das três buscas obtiveram os melhores resultados tanto no MRR quanto no MAP.

Em suma, as redes siamesas em conjunto com o aumento de dados ajudou a gerar melhores *rankings* nos dois experimentos, e incluindo a fusão de *ranking* foram obtidos os melhores resultados.

No Exp1, os dados sintéticos aumentam ainda mais o MRR da busca sstep e da scom, ao contrário do Exp2 em que os mesmo dados sintéticos fizeram a busca scom baixar o valor de MRR. Além disso, a fusão de *ranking* no Exp1 se equiparou à busca scom, ao contrário do Exp2 em que a fusão aumentou todas as pontuações obtidas com as três buscas originais.

No próximo capítulo, fala-se sobre as considerações finais do trabalho.

6

Conclusões

Este trabalho, apresentou uma abordagem para busca de dados de diferentes domínios utilizando redes siamesas por meio de representação vetorial dos dados e por fim, recuperação de informação dos mesmos. A partir dos experimentos realizados constatou-se que a utilização de redes siamesas melhora bastante as métricas de busca de candidatos relevantes para a nossa tarefa de automação de casos de teste. Comparando-a com o *Baseline* usada para fins de validação do método proposto, os resultados foram muito superiores tendo em vista principalmente o aumento da métrica MRR nos dois experimentos propostos.

Utilizar redes siamesas foi de grande valia, pois a arquitetura juntamente com a função de perda usada faz com que a representação de passos e comandos melhorem no espaço vetorial de representação, os *embeddings*. Mais do que isso, ela distorce o espaço vetorial aproximando, no nosso caso, os comandos dos seus passos associados, fazendo com que comandos que antes não poderiam ser encontrados na busca por serem muitas das vezes diferentes, semanticamente, do seu passo associado agora podem ser recomendados como relevantes, por estarem ”próximos” no espaço vetorial de busca.

Como em produção a maioria dos comandos nunca foram automatizados, e portanto não tem um passo associado, esses comandos nunca poderiam ser encontrados em uma busca feita diretamente por passos, neste trabalho é a busca chamada **sstep**.

Portanto utilizou-se técnicas de aumento de dados para que pudesse criar representações sintéticas de passos para serem associadas a esses comandos sem pares. A abordagem para gerar passos sintéticos gerou bons resultados, melhorando muito as nossas buscas, mas mesmo com essa melhora, os métodos baseados em redes siamesas se mantiveram com os melhores resultados.

Por fim, com os *rankings* gerados, ainda aplicou-se técnicas de fusão de *ranking* com o objetivo de gerar um único *ranking* final com os comandos alvo no topo.

Nos testes em ambos experimentos a fusão se saiu bem, porém analisando como um todo pode-se dizer que, não há uma necessidade tão alta de se fundir os *rankings* em produção.É importante ressaltar que, em produção, foram utilizadas muitas outras *features* categóricas referentes as caso de teste como um todo e outras informações referentes ao contexto de teste de *software* para filtrar comandos que não tem relação e para melhorar o *ranking* final, contudo devido a essas informações estarem inseridas no contexto maior do processo de teste e por fugirem do escopo do que ele propõe, elas não puderam ser usadas nesse trabalho.

6.1 Trabalhos Futuros

No decorrer dos experimentos, percebeu-se que algumas abordagens poderiam ser testadas a fim de melhorar os resultados finais como um todo. Uma delas é a realização de testes com novas funções de perda. Atualmente, existem funções de perda que poderiam nos ajudar a melhorar ainda mais a representação dos passos e dos comandos, uma dessas funções é a *Contrastive Loss* que tem a função de aproximar dados de classes iguais e afastar dados de classes diferentes.

Outra ideia seria melhorar a escolha dos negativos da *triplet-loss* usando informações do contexto do caso de teste, para que assim, possamos representar os passos e os comandos com mais precisão em relação ao que realmente acontece produção, ou até mesmo utilizar mais técnicas de escolha dos melhores negativos para serem usados no treinamento da nossa redes siamesa.

Com o crescimento exponencial de modelos pré-treinados em grande escala, outra abordagem que poderia ser testada é um *Fine-Tuning* com os modelos estado-da-arte utilizados atualmente como o *Llamma*, *Falcon*, *CodEdit*, entre outros.

A ideia seria que, como esses modelos foram treinados com muito mais dados que o modelo utilizado neste trabalho, o BLOOM, a representação vetorial dos passos e dos comandos poderia ser melhorada.

REFERÊNCIAS

AI, Q. et al. Information retrieval meets large language models: A strategic report from chinese ir community. *AI Open*, Elsevier, 2023. 44

ALMEIDA, F.; XEXÉO, G. Word embeddings: A survey. *arXiv preprint arXiv:1901.09069*, 2019. 30

ALVIM, L. G. *Busca semântica aplicada à recuperação de informações de contexto histórico*. Tese (Doutorado) — UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO, 2017. 36

AMARICAI, S.; CONSTANTINESCU, R. Designing a software test automation framework. *Informatica Economica*, INFOREC Association, v. 18, n. 1, p. 152, 2014. 46

BAEZA-YATES, R.; RIBEIRO-NETO, B. *Recuperação de Informação-: Conceitos e Tecnologia das Máquinas de Busca*. [S.l.]: Bookman Editora, 2013. 36, 37, 39, 40, 41, 74

BAFNA, P.; PRAMOD, D.; VAIDYA, A. Document clustering: Tf-idf approach. In: IEEE. *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*. [S.l.], 2016. p. 61–66. 30

BATTINA, D. S. Artificial intelligence in software test automation: A systematic literature review. *International Journal of Emerging Technologies and Innovative Research (www. jetir. org| UGC and issn Approved), ISSN*, p. 2349–5162, 2019. 46

BONIFACIO, L. et al. Inpars: Data augmentation for information retrieval using large language models. *arXiv preprint arXiv:2202.05144*, 2022. 32, 45, 47, 48

BROMLEY, J. et al. Signature verification using a"siamese"time delay neural network. *Advances in neural information processing systems*, v. 6, 1993. 42

BURGES, C. J. From ranknet to lambdarank to lambdamart: An overview. *Learning*, v. 11, n. 23-581, p. 81, 2010. 66

CHICCO, D. Siamese neural networks: An overview. *Artificial neural networks*, Springer, p. 73–94, 2021. 53

CRASWELL, N. Mean reciprocal rank. In: ____. *Encyclopedia of Database Systems*. Boston, MA: Springer US, 2009. p. 1703–1703. ISBN 978-0-387-39940-9. Disponível em: <https://doi.org/10.1007/978-0-387-39940-9_488>. 39

DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 29, 53

ESNAASHARI, M.; DAMIA, A. H. Automation of software test data generation using genetic algorithm and reinforcement learning. *Expert Systems with Applications*, Elsevier, v. 183, p. 115446, 2021. 46

FEIFEI, X.; SHUTING, Z.; YU, T. Bert-based siamese network for semantic similarity. In: IOP PUBLISHING. *Journal of Physics: Conference Series*. [S.l.], 2020. v. 1684, n. 1, p. 012074. 44, 47, 48

GAGE, P. A new algorithm for data compression. *C Users Journal*, McPherson, KS: R & D Publications, c1987-1994., v. 12, n. 2, p. 23–38, 1994. 29

GEORGE, K. K.; KUMAR, C. S.; PANDA, A. Cosine distance features for robust speaker verification. In: *Sixteenth annual conference of the international speech communication association*. [S.l.: s.n.], 2015. 37

HOSSEINI-ASL, E.; GUHA, A. Similarity-based text recognition by deeply supervised siamese network. *arXiv preprint arXiv:1511.04397*, 2015. 43

JERONYMO, V. et al. Inpars-v2: Large language models as efficient dataset generators for information retrieval. *arXiv preprint arXiv:2301.01820*, 2023. 45, 47, 48

KLUSCH, M. et al. Semantic web service search: a brief survey. *KI-Künstliche Intelligenz*, Springer, v. 30, p. 139–147, 2016. 37

LI, D. et al. On sampling top-k recommendation evaluation. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. [S.l.: s.n.], 2020. p. 2114–2124. 41

LI, J.; LIU, G. An improved lambdamart algorithm based on the matthew effect. *Mathematical Problems in Engineering*, Hindawi Limited, v. 2018, p. 1–11, 2018. 66

LIU, T.-Y. et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, Now Publishers, Inc., v. 3, n. 3, p. 225–331, 2009. 38

LIU, Y. et al. Topical word embeddings. In: *Twenty-ninth AAAI conference on artificial intelligence*. [S.l.: s.n.], 2015. 30

MARTIN, J. *O que é o Google BERT e como o algoritmo está mudando a forma como buscamos na internet*. 2023. Disponível em: <https://pt.semrush.com/blog/bert/>. 32

MOINDROT, O. *Triplet Loss and Online Triplet Mining in TensorFlow [Blog post]*. 2018. Disponível em: <https://omoindrot.github.io/triplet-loss>. 54

NECULOIU, P.; VERSTEEGH, M.; ROTARU, M. Learning text similarity with siamese recurrent networks. In: *Proceedings of the 1st Workshop on Representation Learning for NLP*. [S.l.: s.n.], 2016. p. 148–157. 44

OPENAI. *GPT-4 Technical Report*. 2023. 31

PARNAMI, A.; LEE, M. Learning from few examples: A summary of approaches to few-shot learning. *arXiv preprint arXiv:2203.04291*, 2022. 60

POKSAPPAIBOON, N. et al. Detecting text semantic similarity by siamese neural networks with malstm in thai language. In: IEEE. *2021 2nd International Conference on Big Data Analytics and Practices (IBDAP)*. [S.l.], 2021. p. 7–11. 42, 47

RADFORD, A. et al. Improving language understanding by generative pre-training. OpenAI, 2018. 31

RAFI, D. M. et al. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In: IEEE. *2012 7th International Workshop on Automation of Software Test (AST)*. [S.l.], 2012. p. 36–42. 46

RENDA, M. E.; STRACCIA, U. Web metasearch: rank vs. score based rank aggregation methods. In: *Proceedings of the 2003 ACM symposium on Applied computing*. [S.l.: s.n.], 2003. p. 841–846. 38

RIPLEY, B. D. et al. The r project in statistical computing. *MSOR Connections. The newsletter of the LTSN Maths, Stats & OR Network*, v. 1, n. 1, p. 23–25, 2001. Disponível em: <https://cran.r-project.org/web/packages/recometrics/vignettes/Evaluating_recommender_systems.html>. 41

RISHITA, M. V. S.; RAJU, M. A.; HARRIS, T. A. Machine translation using natural language processing. In: EDP SCIENCES. *MATEC Web of Conferences*. [S.l.], 2019. v. 277, p. 02004. 30

ROCHA, T. M.; CARVALHO, A. L. D. C. Siameseqat: A semantic context-based duplicate bug report detection using replicated cluster information. *IEEE Access*, IEEE, v. 9, p. 44610–44630, 2021. 35, 43, 47, 48

SCAO, T. L. et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022. 32, 33, 60

SINGHAL, A. et al. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, v. 24, n. 4, p. 35–43, 2001. 37

SNEHA, K.; MALLE, G. M. Research on software testing techniques and software automation testing tools. In: IEEE. *2017 international conference on energy, communication, data analytics and soft computing (ICECDS)*. [S.l.], 2017. p. 77–81. 46

UMAR, M. A.; ZHANFANG, C. A study of automated software testing: Automation tools and frameworks. *International Journal of Computer Science Engineering (IJCSE)*, v. 6, p. 217–225, 2019. 46

VASWANI, A. et al. *Attention Is All You Need*. 2017. 26

VIJI, D.; REVATHY, S. A hybrid approach of weighted fine-tuned bert extraction with deep siamese bi–lstm model for semantic text similarity identification. *Multimedia Tools and Applications*, Springer, v. 81, n. 5, p. 6131–6157, 2022. 44, 47, 48

VIPIN, P.-N. T. M. S. et al. *Introduction to data mining*. 2006. 37

WANG, Y. et al. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, ACM New York, NY, USA, v. 53, n. 3, p. 1–34, 2020. 60

WINKLER, D. et al. A framework for automated testing of automation systems. In: IEEE. *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*. [S.l.], 2010. p. 1–4. 46

ZILIO, L. Terminologia textual e linguística de corpus: estudo em parceria. *EM CORPORA*, p. 184, 2010. 28