

Daniel Baraldi

**Uma Análise sobre o
Impacto do Streaming de
Dados no Contexto da
Visualização**

**Monografia apresentada ao Programa de
Educação Continuada da Escola
Politécnica da Universidade de São Paulo,
para obtenção do título de Especialista,
pelo Programa de Pós-Graduação em
Engenharia de Dados e Big Data.**

**SÃO PAULO
2023**

Daniel Baraldi

**Uma Análise sobre o
Impacto do Streaming de
Dados no Contexto da
Visualização**

**Monografia apresentada ao Programa de
Educação Continuada da Escola
Politécnica da Universidade de São Paulo,
para obtenção do título de Especialista,
pelo Programa de Pós-graduação em
Engenharia de Dados e Big Data.**

**Área de concentração: Tecnologia da
Informação – Engenharia/ Tecnologia/
Gestão**

Orientador: Marcelo Morandini

**SÃO PAULO
2023**

Baraldi, Daniel

Uma Análise sobre o Impacto do Streaming de Dados no Contexto da Visualização / D. Baraldi, M. Morandini -- São Paulo, 2023.

2 p.

Monografia (Especialização em Engenharia de Dados e Big Data) - Escola Politécnica da Universidade de São Paulo. PECE – Programa de Educação Continuada em Engenharia.

1.Visualização de Dados 3.Streaming de Dados I.Universidade de São Paulo. Escola Politécnica. PECE – Programa de Educação Continuada em Engenharia II.t. III.Morandini, Marcelo

FICHA CATALOGRÁFICA

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus por todas as oportunidades profissionais que pude adquirir. Também gostaria de agradecer ao Programa de Educação Continuada da Poli-USP e a todos os seus mestres por ceder espaço ao desenvolvimento deste trabalho e do curso, bem como aos meus familiares por terem me apoiado nessa jornada e minha esposa por estar sempre ao meu lado me suportando.

CURSO ENGENHARIA DE BIG DATA

Coord.: Prof. Solange N. Alves de Souza

Vice-Coord.: Prof. Pedro Luis Pizzigatti Correa

Perspectivas profissionais alcançadas com o curso:

O curso possibilitou que eu pudesse me fixar em uma área nova como Especialista Técnico de Dados e Inteligência Artificial. Através dele, posso discutir sobre diferentes temas de dados com diferentes clientes e profissionais, além de fortalecer meu conhecimento prévio e fomentar a curiosidade existente neste domínio.

RESUMO

O uso de *streaming* de dados e tecnologias nesse sentido tem auxiliado bastante no processamento, análise e uso dos dados em geral e quase em tempo real. Entretanto, um ambiente em que o tempo entre a geração dos dados e o consumo é cada vez menor, pode gerar impactos nas formas de utilização. Dados atualizando constantemente e aspectos relacionados a formato, repetição, forma de geração, entre outros, podem resultar em consequências diretas na visualização, além da própria questão de desempenho gráfico. Considerando esses desafios do *streaming*, esse trabalho tem o objetivo de analisar e medir a visualização de dados e o desempenho nesse cenário a partir de métricas e tecnologias relacionadas ao fluxo de dados, visualização, tempo de atualização e uso de recursos computacionais. Para isso, foi realizada uma implementação de fluxos parecidos, mas variando a tecnologia de processamento (*micro-batches* e processamento em *streaming*) que têm seus desempenhos comparados, bem como a forma de geração dos dados, de forma mais estática ou totalmente aleatória com a tecnologia dos Grandes Modelos de Linguagem. A forma mais estática usa palavras pré-definidas de até vinte produtos, escolhidas de forma aleatória via *script*. Já a geração de dados totalmente aleatória se baseia nas frases geradas pelos Grandes Modelos de Linguagem, e, por esse motivo, são imprevisíveis, devido à sua natureza probabilística, criando diferentes palavras e frases. O fluxo tem início em dados de texto gerados por um *script* com palavras pré-definidas selecionadas aleatoriamente ou a partir da geração de textos de diferentes Grandes Modelos de Linguagem, que passam por tecnologias de *micro-batches*, que são processamentos em lote de pequenos conjuntos de dados que simulam um *streaming*, ou *streaming* de fato, com respectivos agrupamentos de palavras e possíveis tratamentos ao longo do fluxo. Por fim, os dados são visualizados graficamente e quase em tempo real e as métricas são coletadas durante as execuções.

Palavras-chave: *streaming* de dados, *micro-batches*, visualização de dados, fluxo de dados, Grandes Modelos de Linguagem

ABSTRACT

Data streaming and technologies like these help a lot in processing, analysis and data usage in general and in near real time. However, an environment in which time between data creation and data consumption is smaller and smaller can cause impacts in how it is used. Data refreshing very often and other characteristics related to format, duplication, generation method, or others, can result in direct consequences for the visualization, besides the issues about graph performance. Considering challenges like these, this research aims to analyze and measure the data visualization from different metrics and technologies related to data pipeline, visualization, freshness and usage of computational resources. This was executed with the deployment of similar pipelines, but changing the processing technology (micro-batches and streaming processing) that have their performances compared, as long as the way the data was generated, in a more static way or totally randomized with Large Language Models. The more static way is executed using up to twenty predefined products chosen randomly through a script. On the other hand, the totally randomized data generation is based on phrases generated by Large Language Models and, therefore, are unpredictable because of the probabilistic nature of these models, creating different words and phrases. The pipeline starts with text data being generated by a script with predefined words randomly selected or by text generated by different Large Language Models. These data go through micro-batches, that are batch processing of small sets of data and mimic streaming, or real streaming, with respective word grouping and possible treatments through the pipeline. At last, the data are shown graphically and near in real time and the measurements are gathered during the execution.

Keywords: data streaming, micro-batches, data visualization, data pipeline, Large Language Models

LISTA DE FIGURAS

Figura 1 - Análise de Pesquisas na Web por Apache Spark e Apache Flink	13
Figura 2 - Cluster do Apache Kafka	24
Figura 3 - Bibliotecas Necessárias	27
Figura 4 - Bancada Proposta com Script Gerando Dados	30
Figura 5 - Bancada Proposta com Grandes Modelos de Linguagem Gerando Dados	31
Figura 6 - Prompt dos Grandes Modelos de Linguagem	32
Figura 7 - Cluster com Zookeeper e Três Brokers para Apache Kafka no VMware	35
Figura 8 - Visualização Final	39
Figura 9 - Visualização Final	40
Figura 10 - Visualização Final	41
Figura 11 - Visualização Final	42
Figura 12 - Visualização Final	43
Figura 13 - Visualização Final	44
Figura 14 - Visualização Final	46
Figura 15 - Visualização Final	47
Figura 16 - Visualização Final	48
Figura 17 - Visualização Final	49
Figura 18 - Visualização Final	50
Figura 19 - Visualização Final	52
Figura 20 - Tratamento e Conferência dos Dados no PyFlink	54
Figura 21 - Tratamento e Conferência dos Dados no PySpark	54
Figura 22 - Visualização Final	57
Figura 23 - Visualização Final	58
Figura 24 - Visualização Final	59
Figura 25 - Visualização Final	60
Figura 26 - Visualização Final	61
Figura 27 - Visualização Final	62
Figura 28 - Visualização Final	66
Figura 29 - Visualização Final	67
Figura 30 - Visualização Final	68
Figura 31 - Visualização Final	69
Figura 32 - Visualização Final	70
Figura 33 - Visualização Final	71

LISTA DE TABELAS

Tabela 1 - Execução dos Fluxos de Dados com PySpark	16
Tabela 2 - Execução dos Fluxos de Dados com PyFlink	16
Tabela 3 - Amostras por Palavra	39
Tabela 4 - Contagem Total de Amostras com 500 Exemplos	40
Tabela 5 - Contagem Total de Amostras com 50 Iterações de Comentários	41
Tabela 6 - Amostras por Palavra	42
Tabela 7 - Contagem Total de Amostras com 500 Exemplos	43
Tabela 8 - Contagem Total de Amostras com 50 Iterações de Comentários	45
Tabela 9 - Amostras por Palavra	46
Tabela 10 - Contagem Total de Amostras com 500 Exemplos	47
Tabela 11 - Contagem Total de Amostras com 50 Iterações de Comentários	49
Tabela 12 - Amostras por Palavra	50
Tabela 13 - Contagem Total de Amostras com 500 Exemplos	50
Tabela 14 - Contagem Total de Amostras com 50 Iterações de Comentários	52
Tabela 15 - Média e Desvio Padrão do Tempo de Atualização em Segundos	57
Tabela 16 - Média e Desvio Padrão do Tempo de Atualização em Segundos	58
Tabela 17 - Média e Desvio Padrão do Tempo de Atualização em Segundos	59
Tabela 18 - Média e Desvio Padrão do Tempo de Atualização em Segundos	60
Tabela 19 - Média e Desvio Padrão do Tempo de Atualização em Segundos	61
Tabela 20 - Média e Desvio Padrão do Tempo de Atualização em Segundos	62
Tabela 21 - Média dos Tempos de Atualização	63
Tabela 22 - Desvio Padrão dos Tempos de Atualização	63
Tabela 23 - Tempo de Execução Total dos Fluxos	64
Tabela 24 - Tempo de Início de Atualizações Gráficas	64
Tabela 25 - Média e Desvio Padrão de Porcentagem de CPU e RAM	66
Tabela 26 - Média e Desvio Padrão de Porcentagem de CPU e RAM	67
Tabela 27 - Média e Desvio Padrão de Porcentagem de CPU e RAM	68
Tabela 28 - Média e Desvio Padrão de Porcentagem de CPU e RAM	69
Tabela 29 - Média e Desvio Padrão de Porcentagem de CPU e RAM	70
Tabela 30 - Média e Desvio Padrão de Porcentagem de CPU e RAM	71
Tabela 31 - Média de Porcentagens de Uso de CPU	72
Tabela 32 - Desvio Padrão de Porcentagens de Uso de CPU	72
Tabela 33 - Média de Porcentagens de Uso de RAM	72
Tabela 34 - Desvio Padrão de Porcentagens de Uso de RAM	73
Tabela 35 - Tempo Médio de Atualização Gráfica	74
Tabela 36 - Desvio Padrão de Tempo de Atualização Gráfica	74
Tabela 37 - Tempo Total de Execução	74
Tabela 38 - Tempo de Início de Atualizações Gráficas	74
Tabela 39 - Média de Porcentagem de Uso de CPU	75
Tabela 40 - Desvio Padrão de Porcentagem de Uso de CPU	75
Tabela 41 - Média de Porcentagem de Uso de RAM	76
Tabela 42 - Desvio Padrão de Porcentagem de Uso de RAM	76
Tabela 43 - Proporção de Palavras e Frases Incorretas de Acordo com Avaliação Humana	77

SUMÁRIO

1 INTRODUÇÃO	12
1.1 MOTIVAÇÃO	13
1.2 OBJETIVO	14
1.2.1 OBJETIVOS ESPECÍFICOS	14
1.3 JUSTIFICATIVA	15
1.4 METODOLOGIA	15
2 FUNDAMENTAÇÃO TEÓRICA	18
2.1 TECNOLOGIAS DE STREAMING DE DADOS	18
2.2 FATORES HUMANOS E FUNCIONAIS DO STREAMING DE DADOS	18
2.3 MENSAGERIA DE STREAMING E ARQUITETURA ORIENTADA A EVENTOS	19
2.4 FLUXO DE DADOS	19
2.5 LINGUAGENS DE PROGRAMAÇÃO	20
2.6 GERENCIADOR DE INTEGRAÇÃO DE SOFTWARES E BIBLIOTECAS	20
2.7 VISUALIZAÇÃO DE DADOS	20
2.8 MODELOS DE LINGUAGEM GRANDE E MODELOS FUNDACIONAIS	21
3 ANÁLISE DAS TECNOLOGIAS UTILIZADAS	22
3.1 WINDOWS E GOOGLE SHEETS	22
3.2 VMWARE WORKSTATION PLAYER 17	22
3.3 APACHE KAFKA	22
3.4 APACHE SPARK	24
3.5 APACHE FLINK	25
3.6 PYTHON E MINICONDA	26
3.7 MATPLOTLIB	27
3.8 LLAMA-CPP-PYTHON, GRANDES MODELOS DE LINGUAGEM E HUGGING FACE	28
3.9 BIBLIOTECAS PARA MÉTRICAS	29
3.10 BANCADA EXPERIMENTAL E CONFIGURAÇÕES GERAIS	30
4 RESULTADOS	35
4.1 RESULTADO DA VISUALIZAÇÃO	38
4.1.1 SEM TRATAMENTO	38
4.1.1.1 PYSPARK	38
4.1.1.1.1 TRÊS COLUNAS/PRODUTOS E SEM TRATAMENTO (500 EXEMPLOS)	39
4.1.1.1.2 VINTE COLUNAS/PRODUTOS E SEM TRATAMENTO (500 EXEMPLOS)	40
4.1.1.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF E SEM TRATAMENTO (50 GERAÇÕES DE TEXTO)	41
4.1.1.2 PYFLINK	42
4.1.2.1.1 TRÊS COLUNAS/PRODUTOS E SEM TRATAMENTO (500 EXEMPLOS)	42
4.1.2.1.2 VINTE COLUNAS/PRODUTOS SEM TRATAMENTO (500 EXEMPLOS)	43
4.1.2.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF E SEM TRATAMENTO (50 GERAÇÕES DE TEXTO)	44
4.1.2 COM TRATAMENTO	45
4.1.2.1 PYSPARK	46
4.1.2.1.1 TRÊS COLUNAS/PRODUTOS E COM TRATAMENTO (500 EXEMPLOS)	46
4.1.2.1.2 VINTE COLUNAS/PRODUTOS E COM TRATAMENTO (500 EXEMPLOS)	47
4.1.2.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF COM TRATAMENTO (50 GERAÇÕES DE TEXTO)	48
4.1.2.2 PYFLINK	49

4.1.2.2.1 TRÊS COLUNAS/PRODUTOS COM TRATAMENTO (500 EXEMPLOS)	49
4.1.2.2.2 VINTE COLUNAS/PRODUTOS E COM TRATAMENTO (500 EXEMPLOS)	50
4.1.2.2.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF E COM TRATAMENTO (50 GERAÇÕES DE TEXTO)	52
4.1.2 ANÁLISE DA VISUALIZAÇÃO	52
4.2 RESULTADO DO TEMPO DE ATUALIZAÇÃO	56
4.2.1.1 PYSPARK	56
4.2.1.1.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)	57
4.2.1.1.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)	58
4.2.1.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)	59
4.2.1.2 PYFLINK	60
4.2.1.2.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)	60
4.2.1.2.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)	61
4.2.1.2.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)	62
4.2.1.3 ANÁLISE DO TEMPO DE ATUALIZAÇÃO	63
4.2.2 CONSUMO DE CPU E MEMÓRIA RAM	65
4.2.2.1 PYSPARK	66
4.2.2.1.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)	66
4.2.2.1.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)	67
4.2.2.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)	68
4.2.2.2 PYFLINK	69
4.2.2.2.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)	69
4.2.2.2.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)	70
4.2.2.2.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)	71
4.2.2.3 ANÁLISE DE USO DOS RECURSOS COMPUTACIONAIS	72
4.3 RESULTADO DOS GRANDES MODELOS DE LINGUAGEM	74
5 CONCLUSÃO	78
6 CONTRIBUIÇÕES DO TRABALHO	80
7 TRABALHOS FUTUROS	81
8 REFERÊNCIAS BIBLIOGRÁFICAS	82
APÊNDICE A - SCRIPTS EXECUTADOS E GITHUB COM MATERIAIS	90
APÊNDICE B - RESULTADOS DAS VISUALIZAÇÕES E TABELAS	103

1 INTRODUÇÃO

Uma das questões que têm se intensificado nos últimos anos é o uso de *streaming* de dados. Isso se deve ao rápido avanço das tecnologias e também da necessidade cada vez mais frequente do consumo imediato dos dados para que ações sejam tomadas no menor prazo possível, evitando assim prejuízos ou antevendo oportunidades.

Porém, com o uso de tecnologias atualmente disponíveis, existe também a necessidade de se verificar ou medir os comportamentos dessas diferentes abordagens tecnológicas para se obter o melhor resultado em um cenário de um fluxo completo de visualização de dados.

Nesse sentido, este trabalho procurou apresentar análises acerca de um fluxo pré-definido de dados, com ligeiras variações na bancada e no tipo de processamento de dados, bem como diferentes formas de visualização.

Dentre as tecnologias utilizadas, foi empregado um *script* que selecionou palavras pré-definidas aleatoriamente, além de Grandes Modelos de Linguagem para a geração de palavras de forma totalmente aleatória, de forma a simular cenários de dados diferentes e crescentes. Para os modelos de inteligência artificial generativa que fazem as gerações de textos, foram selecionados três tipos: TheBloke/Llama-2-7B-Chat-GGUF, TheBloke/Llama-2-13B-chat-GGUF e TheBloke/zephyr-7B-beta-GGUF. Todos esses modelos foram utilizados localmente em formato GGUF a partir do *framework* llama-cpp para Python, de forma a permitir essa execução local com uso da placa de vídeo (GERGANOV, 2023). Além disso, todos os três modelos foram utilizados no formato quantizado de 4 bits de forma a reduzir o tamanho das redes neurais, melhorando o tempo de processamento (NEVES, 2023). Essas tecnologias realizaram a seleção ou criação de palavras, que foram então contabilizadas.

Para o processamento dos dados, foram escolhidas abordagens de *micro-batches* (Apache Spark) e *streaming* de fato (Apache Flink), que realizam processos de transformação, extração e carga. Além disso, no fluxo de dados, também foi utilizado

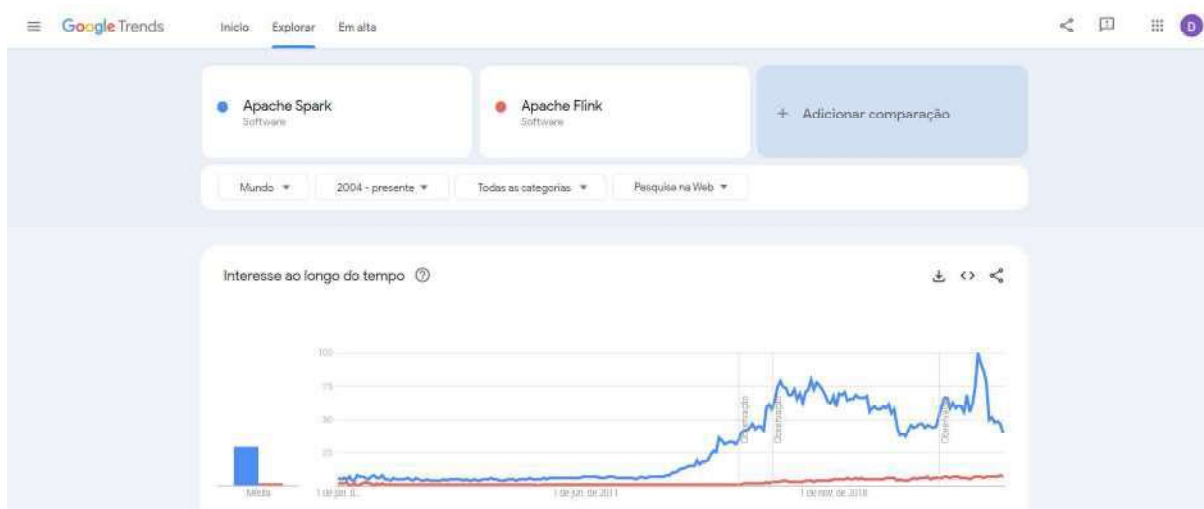
um modelo de inteligência artificial generativa para realizar a geração de dados, que no caso são palavras que estão sendo contabilizadas.

Então, o fluxo de dados utilizado neste trabalho basicamente é um caminho com geração, processamento, transformação e visualização por onde os dados percorrem. Esse caminho é representado por uma bancada que tem seus componentes variados levemente de forma a analisar possíveis impactos nas métricas de tempo de execução, uso de recurso computacional, bem como percepções mais subjetivas de visualização de dados de forma a obter boas práticas.

1.1 MOTIVAÇÃO

Como recentemente os *micro-batches* na tecnologia do Apache Spark são mais populares quando comparados com outras como o processamento em *streaming* de fato, como é o caso do Apache Flink (conforme a Figura 1 do Google Trends), surge a necessidade de comparar essas abordagens tecnológicas para validar se existe um ganho de desempenho ao usar um ou outro tipo de abordagem na visualização de dados.

Figura 1 - Análise de Pesquisas na Web por Apache Spark e Apache Flink



Fonte: Google Trends (<https://www.google.com/trends>)¹

¹ Disponível em:

<https://trends.google.com/trends/explore?date=all&q=%2Fm%2F0ndhxqz,%2Fg%2F11btzy2gtf&hl=p>
t>. Acesso em: 29 de dez. de 2023.

Comparativos desse tipo são usados em pesquisas distintas com considerações em relação à latência, armazenamento, e *framework* de processamento (AKRAM e col., 2019). Além disso, visualizações em tempo real são colocadas em situações distintas de processamentos em lote, considerando até mesmo fatores humanos no uso (Dasgupta e col. 2018).

No trabalho em questão, a comparação foi feita através da percepção humana diante das visualizações gráficas, de forma mais subjetiva, mas também a partir de métricas objetivas como tempo de atualização gráfica e uso de recursos computacionais.

1.2 OBJETIVO

Esse trabalho teve o objetivo de verificar o desempenho de tecnologias de *streaming* de dados e o resultado na visualização de dados a partir do desafio relacionado a atualizações na visualização e outras características dos dados, como formato, acúmulo, repetição e forma de geração com e sem o uso de Grandes Modelos de Linguagem.

1.2.1 OBJETIVOS ESPECÍFICOS

Dentre os objetivos específicos que compõem o objetivo geral, a primeira necessidade foi a realização de um fluxo de dados composto por um *script* ou por Grandes Modelos de Linguagem para geração dos dados que serão processados por diferentes tecnologias.

O primeiro objetivo foi então a consolidação de boas práticas de uso da visualização de dados no contexto de *streaming* de dados e verificação de diferenças subjetivas entre as duas tecnologias principais utilizadas (*micro-batches* e *streaming* de fato), bem como definição de boas práticas da visualização de dados no contexto de atualizações quase em tempo real.

O segundo objetivo foi a comparação entre duas abordagens tecnológicas (*micro-batches* e *streaming* de fato) no contexto do processamento para

visualização de dados através de métricas objetivas como tempo de atualização e consumo de recursos computacionais.

O terceiro e último objetivo foi a comparação entre os diferentes Grandes Modelos de Linguagem no contexto de *streaming*.

Atendendo a todos esses objetivos específicos, foi possível atender ao objetivo geral deste trabalho.

1.3 JUSTIFICATIVA

Com essas diferentes abordagens (*micro-batches* e processamento em *streaming*), foi necessário analisar como cada uma se comporta em um contexto de visualização de dados e como cada uma impacta o resultado final desse fluxo de dados para que seja possível verificar quais as vantagens e desvantagens relacionadas a desempenho e uso em um contexto desse tipo.

Atualmente, a tecnologia amplamente utilizada e bastante popular é a de *micro-batches*. Com uma análise deste tipo, é possível perceber como a simples alteração de *micro-batches* para *streaming* ou o inverso poderia fornecer ganhos expressivos em fluxos de dados que não apenas possuem a visualização de dados como fim, mas também aqueles que possuem modelos de aprendizado de máquina ou simples registros em sistemas que precisam ocorrer em períodos curtos de tempo que se classificam como tempo real ou quase tempo real.

1.4 METODOLOGIA

Para a realização deste trabalho, foram compostos dois tipos de fluxos de dados, com cinco fluxos em cada tipo. O primeiro tipo é usando uma abordagem de *micro-batches* e o segundo com processamento de *streaming*. Esses fluxos de dados vão desde a geração dos dados até a visualização, com diferentes processamentos aplicados a esses dados e diferentes formas de geração.

Foram executados 40 fluxos de dados, divididos da seguinte forma:

Tabela 1 - Execução dos Fluxos de Dados com PySpark

Tipo 1 (Apache Spark/micro-batches)			
Resultado da Visualização		Resultado do Tempo de Atualização	Resultado do Consumo de Recursos Computacionais
Sem tratamento gráfico	Com tratamento gráfico	Com tratamento gráfico	Com tratamento gráfico
Geração de 3 Colunas/Palavras	Geração de 3 Colunas/Palavras	Geração de 3 Colunas/Palavras	Geração de 3 Colunas/Palavras
Geração de 20 Colunas/Palavras	Geração de 20 Colunas/Palavras	Geração de 20 Colunas/Palavras	Geração de 20 Colunas/Palavras
Modelo de Linguagem Grande (llama 7b)	Modelo de Linguagem Grande (llama 7b)	Modelo de Linguagem Grande (llama 7b)	Modelo de Linguagem Grande (llama 7b)
Modelo de Linguagem Grande (llama 13b)	Modelo de Linguagem Grande (llama 13b)	Modelo de Linguagem Grande (llama 13b)	Modelo de Linguagem Grande (llama 13b)
Modelo de Linguagem Grande (zephyr 7b)	Modelo de Linguagem Grande (zephyr 7b)	Modelo de Linguagem Grande (zephyr 7b)	Modelo de Linguagem Grande (zephyr 7b)

Tabela 2 - Execução dos Fluxos de Dados com PyFlink

Tipo 2 (Apache Flink/streaming)			
Resultado da Visualização		Resultado do Tempo de Atualização	Resultado do Consumo de Recursos Computacionais
Sem tratamento gráfico	Com tratamento gráfico	Com tratamento gráfico	Com tratamento gráfico
Geração de 3 Colunas/Palavras	Geração de 3 Colunas/Palavras	Geração de 3 Colunas/Palavras	Geração de 3 Colunas/Palavras
Geração de 20 Colunas/Palavras	Geração de 20 Colunas/Palavras	Geração de 20 Colunas/Palavras	Geração de 20 Colunas/Palavras
Modelo de Linguagem Grande	Modelo de Linguagem Grande	Modelo de Linguagem Grande	Modelo de Linguagem Grande

(llama 7b)	(llama 7b)	(llama 7b)	(llama 7b)
Modelo de Linguagem Grande (llama 13b)	Modelo de Linguagem Grande (llama 13b)	Modelo de Linguagem Grande (llama 13b)	Modelo de Linguagem Grande (llama 13b)
Modelo de Linguagem Grande (zephyr 7b)	Modelo de Linguagem Grande (zephyr 7b)	Modelo de Linguagem Grande (zephyr 7b)	Modelo de Linguagem Grande (zephyr 7b)

O fluxo de dados foi construído com tecnologias específicas para o contexto e de forma local, sendo que para cada execução, foi realizada a exclusão de arquivos criados durante os processos para evitar diferenças nas métricas. Além disso, foram excluídos também outros registros dos dados/mensagens utilizados.

Para a elaboração das boas práticas de visualização de dados em *streaming* foram executados fluxos sem nenhum tipo de tratamento gráfico e fluxos com tratamentos gráficos de forma a melhorar a visualização final. Isso foi feito para que fosse possível analisar as melhorias com tratamentos realizados na visualização e também o resultado que isso causa na percepção dos dados.

Já para o comparativo entre as tecnologias de *micro-batches* e *streaming*, foram coletadas métricas como tempo de atualização gráfica e consumo de recurso de CPU e memória RAM da máquina utilizada. O comparativo foi então realizado com as diversas abordagens de visualização, desde a geração de dados estática (com palavras pré-definidas selecionadas via *script* aleatoriamente e que são contabilizadas) até o uso dos Grandes Modelos de Linguagem (que geram frases com palavras que são contabilizadas).

Por fim, são elaboradas métricas como média e desvio padrão para os respectivos fluxos de forma a validar como essas abordagens se comportam nos cenários de visualização de dados, sendo que todos os *scripts* utilizados, bem como demais informações referentes à execução do trabalho se encontram no Apêndice A, com todos os resultados dessas métricas e demais resultados no Apêndice B.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta características e conceitos acerca dos temas envolvidos neste trabalho, tais como: Tecnologias de *Streaming* de Dados, Fatores Humanos e Funcionais do *Streaming* de Dados, Mensageria de *Streaming*, Fluxo de Dados, Linguagens de Programação, Grandes Modelos de Linguagem, Gerenciador de Integração de *Softwares* e Visualização de Dados.

2.1 TECNOLOGIAS DE *STREAMING* DE DADOS

Diversos tipos de tecnologias são usados para realizar o que é conhecido como *streaming* de dados. Entre elas, é possível listar Spark, Storm, Samza e Flink, sendo que cada tecnologia tem considerações em relação a latência, armazenamento, e *framework* de processamento (AKRAM e col., 2019). Portanto, tecnologias desse tipo e suas considerações podem ser utilizadas para a criação de fluxos de dados. Avaliando-se as limitações e o objetivo de cada opção, duas tecnologias são eleitas para a realização de parte do fluxo (Apache Spark e Apache Flink), usando respectivamente *micro-batches* e processamento *streaming*.

O Apache Spark é comumente utilizado em cenários de *streaming* de dados e por esse motivo foi selecionado para esse estudo. Por mais que seja considerado para o uso em *streaming*, o Apache Spark na realidade usa uma abordagem que simula um processamento em *streaming* através de *micro-batches*. Mesmo assim, a tecnologia se popularizou (ILYUKHA, [s.d.]) com o intuito de atender a cenários de tempo real.

Para a comparação, então, foi selecionado o Apache Flink, que é uma tecnologia mais atual e que utiliza *streaming* efetivamente.

2.2 FATORES HUMANOS E FUNCIONAIS DO *STREAMING* DE DADOS

Além de fatores como desempenho, tempo de atualização, formato e repetição de dados, existem fatores relacionados à própria visualização. Visualizações que sejam mais consistentes com a mudança constante e com o acúmulo de dados são preferenciais em um cenário de *streaming* (Dasgupta e col. 2018). Por esse motivo,

visualizações adequadas são necessárias para a existência de um entendimento humano correto dos dados. Para cada cenário, há então uma visualização mais adequada e essa visualização pode ser usada para esse trabalho em questão de forma a consolidar boas práticas no fluxo de dados específico.

2.3 MENSAGERIA DE *STREAMING* E ARQUITETURA ORIENTADA A EVENTOS

A mensageria de *streaming* é algo que deve ser levado em consideração em cenários desse tipo. Isso porque é necessário entender e planejar o que deve ser feito em relação a múltiplas mensagens atualizando e a constante mudança dos dados ao longo do tempo são algumas das questões a serem consideradas (Dasgupta e col. 2018).

Neste trabalho, foi usado um cluster de Apache Kafka, que trabalha com três nós e realiza a produção de mensagens e também o recebimento. No caso de uma parada ou algo similar, as mensagens são enfileiradas e depois consumidas quando possível (“Apache Kafka”, [s.d.]).

Esse tipo de abordagem é utilizado em arquiteturas orientadas a eventos, onde existem produtores, consumidores e mensagens. Esse tipo de arquitetura suporta o trabalho em tempo real, sendo que o Apache Kafka se apresenta como um componente possível, de acordo com a RED HAT (2019).

2.4 FLUXO DE DADOS

Fluxos de dados são construções de arquiteturas de dados que seguem um caminho específico até que o dado seja consumido. Durante esse caminho, podem ocorrer extrações, transformações e cargas de dados. E, por fim, o dado pode ser consumido gerando uma visualização ou alimentando um modelo de aprendizado de máquina, por exemplo, de acordo com a definição da CONFLUENT ([s.d.]).

Neste trabalho, foi utilizado um fluxo que passa por diferentes tecnologias e chega até uma visualização. Com isso, foi possível perceber como a troca de uma tecnologia por outra influencia no impacto do fluxo como um todo.

2.5 LINGUAGENS DE PROGRAMAÇÃO

O trabalho foi desenvolvido usando a mesma linguagem de programação Python. Foi eleita para esse trabalho devido à sua capacidade de ter módulos que suportam extensões escritos em qualquer linguagem compilada (“CONCEPTS OF PROGRAMMING LANGUAGES TENTH EDITION”, [s.d.]; SEBESTA, 2012), além da facilidade de uso .

De forma a evitar diferenças causadas por tempo de compilação e execução causado por diferentes linguagens, o Python foi utilizado em todas as etapas necessárias como padrão.

2.6 GERENCIADOR DE INTEGRAÇÃO DE *SOFTWARES* E BIBLIOTECAS

Além do Python, foi utilizado um gerenciador de integração de *softwares* juntamente com bibliotecas específicas para a configuração do fluxo de dados e execução do código.

Para esse gerenciamento, a tecnologia utilizada foi o miniconda, que é um instalador mínimo do conda. É uma versão menor do Anaconda que inclui Python, conda (gerenciador de bibliotecas) e algumas bibliotecas específicas (“Miniconda — miniconda documentation”, [s.d.]).

2.7 VISUALIZAÇÃO DE DADOS

A etapa de visualização também foi uma etapa extremamente importante para esse trabalho, já que é a partir dela que foi feito todo o estudo desde a percepção até a extração de métricas de atualização ou uso de recurso computacional. Além disso, a visualização de dados foi fundamental para que fossem estabelecidas boas práticas do uso em cenário de *streaming*.

2.8 MODELOS DE LINGUAGEM GRANDE E MODELOS FUNDACIONAIS

Uma das tecnologias utilizadas no trabalho foi de Grandes Modelos de Linguagem. De acordo com a IBM, Grandes Modelos de Linguagem são uma classe de modelos fundacionais que foram treinados em grandes quantidades de dados para obterem capacidades necessárias para direcionar múltiplos casos de uso (IBM, [s.d.]). Modelos fundacionais, por sua vez, são descritos pela IBM, com base na primeira popularização do termo *pelo Stanford Institute for Human-Centered Artificial Intelligence*, como modelos treinados em um conjunto de dados não rotulados e que podem ser usados para diferentes tarefas (IBM, 2021). A IBM também descreve que os modelos fundacionais podem ser usados como fundação de diversas aplicações de inteligência artificial a partir da aplicação de aprendizado auto-supervisionado e transferência de aprendizado para aplicar o que as informações de uma situação para outra (IBM, 2021).

Esses Grandes Modelos de Linguagem foram adicionados à bancada devido à sua criação dinâmica, contínua e, de certa forma, imprevisível, já que modelos desse tipo realizam a geração de textos com base na probabilidade de palavras em textos.

3 ANÁLISE DAS TECNOLOGIAS UTILIZADAS

Foram utilizadas diversas tecnologias específicas para cada etapa da construção e análise do fluxo de dados. Com isso, é necessário descrever cada uma delas e suas aplicações nas bancadas finais.

3.1 WINDOWS E GOOGLE SHEETS

Para esse trabalho também foi utilizada a máquina Lenovo Ideapad Gaming 3i, intel core i7-10750H, com uma placa Nvidia Geforce GTX 1650, com 4GB de RAM dedicada e 16GB de memória RAM para a máquina.

Como todo o trabalho foi desenvolvido localmente, as configurações da máquina são fundamentais para o bom processamento de todo o fluxo de dados.

Além disso, a análise das métricas, com médias e desvio padrão foram realizadas utilizando o Google Sheets.

3.2 VMWARE WORKSTATION PLAYER 17

VMware Workstation Player 17 é um software que trabalha com virtualização de máquinas de diferentes sistemas operacionais empregando imagens desses sistemas (VMWARE, 2019). Neste trabalho, foi utilizada a imagem de um Ubuntu 22.04 (UBUNTU, 2019), com as devidas atualizações de sistema operacional e instalando o Apache Kafka nessa máquina com três brokers conforme indicado por AMARAL (2023), seguindo o passo a passo do curso prático. Todo esse ambiente foi executado na máquina Windows.

3.3 APACHE KAFKA

Segundo a própria documentação do Apache Kafka, ele é uma tecnologia de mensageria que permite o enfileiramento de mensagens. Para isso, usa alguns itens como tópicos, brokers, consumidores, produtores e eventos (“Apache Kafka”, [s.d.]).

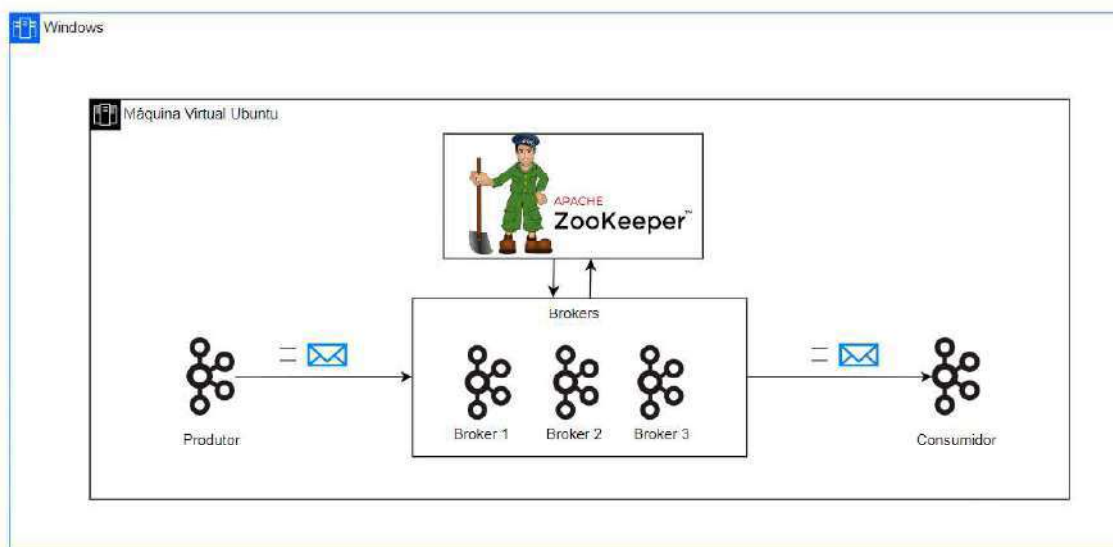
Tópicos são similares a pastas, onde os eventos (mensagens ou registros) são armazenados. Brokers são servidores responsáveis por escrever as mensagens dos Produtores. E os Produtores, por sua vez, são responsáveis por criar as mensagens que serão direcionadas aos Consumidores (“Apache Kafka”, [s.d.]).

Além desses componentes, o principal é um servidor bootstrap, que é criado com o auxílio do Apache Zookeeper, responsável pelo gerenciamento dos brokers (“Apache Kafka”, [s.d.]; TEAM, 2022)

Nesse trabalho, é utilizado um cluster de três brokers Apache Kafka em uma máquina virtual Ubuntu 22.04 sendo executada a partir do software VMware Workstation Player 17. Nessa mesma máquina também é executado o Apache Zookeeper para realizar o gerenciamento e coordenação dos brokers. Os produtores e consumidores, por sua vez, são executados a partir da API do Apache Kafka para Python, no ambiente do Windows que hospeda a máquina virtual, conforme indicado por AMARAL (2023).

Em resumo, os componentes do cluster de Apache Kafka podem ser descritos com através da Figura 2.

Figura 2 - Cluster do Apache Kafka



Fonte: criação do autor².

Como o trabalho foi realizado usando Python, além do cluster, foi necessário instalar as devidas bibliotecas para que fosse possível utilizar o Apache Kafka diretamente dos *scripts* em Python. Com isso, foi utilizada a versão do cliente para Python do Kafka (POWERS; ARTHUR, 2016).

Além disso, o consumo das mensagens em Python foi realizada iterando no consumidor de mensagens, como indicado em “Kafka Tutorial in Python” ([s.d.]), além do uso dessa referência para auxílio de demais construções em geral da interação com o Apache Kafka via Python.

3.4 APACHE SPARK

O Apache Spark, de acordo com sua documentação, é uma *engine* utilizada para executar o processamento de dados. Pode ser utilizado para diversos processos em *streaming* e até mesmo capacidades de aprendizado de máquina (“Overview - Spark 3.0.0 Documentation”, [s.d.]).

² compilação do autor utilizando a ferramenta draw.io a partir de imagens dos sites Wikipédia e Apache Kafka. Disponíveis em: <https://en.wikipedia.org/wiki/Apache_ZooKeeper> e <<https://kafka.apache.org/documentation/#zk>>. Acesso em 26 dez. 2023.

Para esse trabalho, o Apache Spark foi utilizado através de sua API para Python, denominada PySpark, na versão 3.4.1 (“Welcome to Spark Python API Docs! — PySpark 2.4.5 Documentation”, 2023). Com isso, é possível submeter jobs que realizam o processamento de dados em tempo real ou quase em tempo real. Funções dessa API realizam processos de SQL, como agrupamentos, renomeações e seleções de dados. Além disso, foi também utilizada sua capacidade de *streaming*, denominada Spark Structured Streaming, com o Apache Kafka como *source* e *sink*, seguindo a documentação e orientação da comunidade Spark (“Structured Streaming + Kafka Integration Guide (Kafka Broker Version 0.10.0 or higher) - Spark 2.4.5 Documentation”, [s.d.]) e, principalmente, seguindo as instruções de acordo com PANDEY; LEARNING JOURNAL (2023) e também AMARAL (2023b). Ou seja, o Apache Kafka é tanto destino quanto fonte dos dados e esses dados são processados pelo PySpark, de forma a terem agregações e agrupamentos de palavras no contexto em questão.

Para a integração com o Apache Kafka, foi utilizado um pacote específico denominado de “org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0” de acordo com orientações da documentação (“Structured Streaming + Kafka Integration Guide (Kafka Broker Version 0.10.0 or higher) - Spark 2.4.5 Documentation”, [s.d.]) e o artigo de PEDRO (2022). No caso deste trabalho, foi usada a versão descrita por ser recente e estável. O *job* com o *script* foi então submetido, iniciando a execução do processamento.

O uso do PySpark está atrelado ao processamento em *streaming* simulado, com uso de *micro-batches* (“Structured Streaming Programming Guide - Spark 2.4.5 Documentation”, [s.d.]). Com isso, há uma diferença com tecnologias em *streaming* de fato e essa comparação foi verificada como um dos objetivos deste trabalho.

3.5 APACHE FLINK

Segundo sua documentação, o Apache Flink é um framework e engine de processamento de dados. Ele funciona de forma a realizar processamentos em *streaming*, diferente da abordagem de *micro-batches* do PySpark que apenas simula

um *streaming*. Com isso, é possível realizar os mesmos processos executados pelo PySpark.

Para esse trabalho, o Apache Flink foi utilizado através de uma API para o Python, denominada PyFlink. Assim, é possível executar scripts Python com *jobs* que usam o Apache Kafka como *source* e *sink*, assim como é feito com o PySpark. Também foram utilizadas funções SQL para os processamentos de dados, que são executados na configuração de *streaming*.

Para a criação desse *script* de execução, foram utilizadas a documentação padrão para a versão 1.18 (“Welcome to Flink Python Docs! — PyFlink 1.18.dev0 Documentation”, [s.d.]), bem como as instruções práticas fornecidas por BIG DATA LANDSCAPE (2023).

Foi necessário também o uso de um arquivo do tipo jar para a integração entre Apache Kafka e a API do Apache Flink para Python (também denominada PyFlink), que pode ser encontrado no repositório Maven (“ Flink : Connectors : SQL : Kafka» 3.0.1-1.18”, 2023), de acordo com a documentação em questão da API, que indica como deve ser realizada a integração com conectores (“Dependency Management”, [s.d.]).

Assim, o *script* foi executado de forma similar a demais *scripts* Python, sem necessidade de ser submetido como um *job* (como foi necessário no caso do PySpark).

3.6 PYTHON E MINICONDA

Como linguagem de programação, foi utilizada a versão 3.11.5 do Python (“Python Release Python 3.11.5”, 2023), com Miniconda, para executar os *scripts* do produtor e consumidor Kafka via API e os *scripts* de PySpark ou PyFlink via API, bem como o módulos de Grandes Modelos de Linguagem.

Além disso, o Python foi escolhido como padrão de forma a evitar possíveis diferenças entre linguagens de programação e suas características específicas.

Miniconda, segundo sua documentação, é um instalador do gerenciador de pacotes conda, originário do Anaconda (“Miniconda — Miniconda Documentation”, [s.d.]). Sua escolha é devido às orientações da própria documentação do PyTorch recomendando o uso (“Start Locally”, [s.d.]).

Para a execução dos *scripts* específicos, foi necessário realizar a instalação de diversos pacotes ou módulos que possuem versões em Python. Os pacotes necessários foram indicados na Figura 3, através do método descrito e fornecido por KRAVCENKO (2023), com exceção do llama-cpp-python, que precisa de uma instalação específica para executar com GPU.

Figura 3 - Bibliotecas Necessárias

```
C: > Users > baral > OneDrive > Documents > Engenharia de Dados - USP > Monografia > requirements.txt
1  apache_flink==1.18.0
2  apache_flink_libraries==1.18.0
3  fairscale==0.4.13
4  fire==0.5.0
5  kafka_python==2.0.2
6  matplotlib==3.8.0
7  numba==0.58.0
8  numpy==1.24.3
9  numpy==1.24.4
10 psutil==5.9.0
11 pyspark==3.4.1
12 sentencepiece==0.1.99
13 setuptools==68.0.0
14 torch==2.0.1+cu117
```

Fonte: criada pelo autor

3.7 MATPLOTLIB

Além das tecnologias citadas anteriormente, foi utilizada também a biblioteca Matplotlib para a construção gráfica. Por sua característica de personalização a nível de necessidade do usuário, a biblioteca permite a construção dos gráficos de forma mais livre e dinâmica, com uma série de configurações mais personalizadas em relação aos eixos da visualização, escalas, tipos de gráficos, espaçamentos, cores, entre outras, de acordo com as necessidades, o que contribui para que a análise entre as duas tecnologias de processamento de dados e as boas práticas de visualização sejam feitas de forma assertiva.

Construções de gráficos de barras e outras configurações utilizadas foram realizadas seguindo a documentação em “Matplotlib: Python Plotting — Matplotlib 3.1.1 Documentation” (2012).

3.8 LLAMA-CPP-PYTHON, GRANDES MODELOS DE LINGUAGEM E HUGGING FACE

Para a parte de Grandes Modelos de Linguagem, foi necessário primeiro realizar a instalação do llama-cpp-python com a especificação de usar a GPU da Nvidia presente na máquina Windows, de forma a acelerar o resultado dos modelos, já que o llama-cpp-python permite a execução de modelos desse tipo de forma local e eficiente (GERGANOV, 2023). Para isso, foram seguidas documentações específicas com passos muito bem estabelecidos. De acordo com META RESEARCH (2023), é necessário um ambiente com PyTorch/CUDA de forma a executar um modelo como o Llama 2. Além disso, instruções do PYAJUDEME (2023), descrevem a necessidade de um PyTorch instalado e compatível com a placa Nvidia em questão com a respectiva CUDA. A partir da documentação do PyTorch (“PyTorch”, [s.d.]), foi feita a instalação da versão correta e foram feitas as devidas configurações do llama-cpp-python seguindo as instruções de PYAJUDEME (2023), META RESEARCH (2023), ANDREI (2023) e GERGANOV (2023).

Os modelos utilizados foram o TheBloke/Llama-2-7B-Chat-GGUF (JOBINS, 2023b), que possui 7 bilhões de parâmetros, TheBloke/Llama-2-13B-chat-GGUF (JOBINS, 2023a), com 13 bilhões de parâmetros, e TheBloke/zephyr-7B-beta-GGUF (JOBINS, 2023c), com 7 bilhões de parâmetros também. Os modelos foram escolhidos de acordo com *benchmarks*, quantidades de parâmetros e capacidade de execução na máquina local. Todos eles foram obtidos a partir do Hugging Face e da página TheBloke (JOBINS, [s.d.]), que disponibiliza modelos no formato GGUF.

Os modelos em questão são do tipo Q4_K - "type-1" (JOBINS, 2023a). Isso significa que houve uma redução no tamanho dos modelos de forma a acelerar sua execução, mas com uma possível perda de acurácia (NEVES, 2023). Essa diminuição de tamanho pode afetar a acurácia e a precisão dos modelos, porém,

para o trabalho, essa técnica se mostra bastante interessante uma vez que aumenta o desempenho do modelo e possibilita que ele seja executado em máquinas mais simples, como é o caso da utilizada.

Esse ganho de desempenho possibilita que sejam utilizadas características de gerações de palavras para avaliar como as tecnologias de *micro-batches* e *streaming* se comportam também nesse cenário.

3.9 BIBLIOTECAS PARA MÉTRICAS

Para coletar métricas dos fluxos de dados, foi necessário também utilizar duas bibliotecas Python: `datetime` e `psutil`.

A biblioteca `datetime` foi usada para registrar os tempos de execução e de atualização gráfica, para que fosse possível realizar o comparativo.

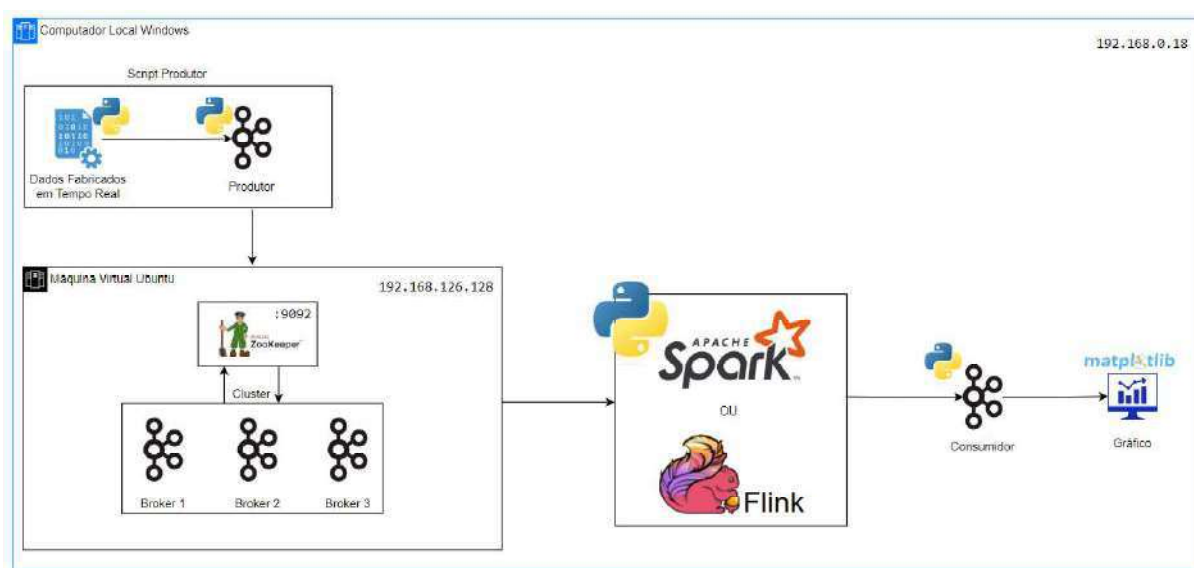
A biblioteca `psutil`, por sua vez, foi responsável por fornecer o consumo de CPU e memória RAM da máquina em questão enquanto o *script* estava sendo executado.

Foram definidos então trechos dos *scripts* determinados a coletar essas informações, seguindo as documentações PYTHON SOFTWARE FOUNDATION ([s.d.]) e RATED_R_SUNDRAM (2023), para os tempos, e RODOLA ([s.d.]), RUPANISWEETY (2021) e GOYAL (2022) para CPU e RAM.

3.10 BANCADA EXPERIMENTAL E CONFIGURAÇÕES GERAIS

As bancadas experimentais propostas para este trabalho são então demonstradas na Figura 4 e na Figura 5. Para essas bancadas, foram utilizadas todas as tecnologias descritas anteriormente.

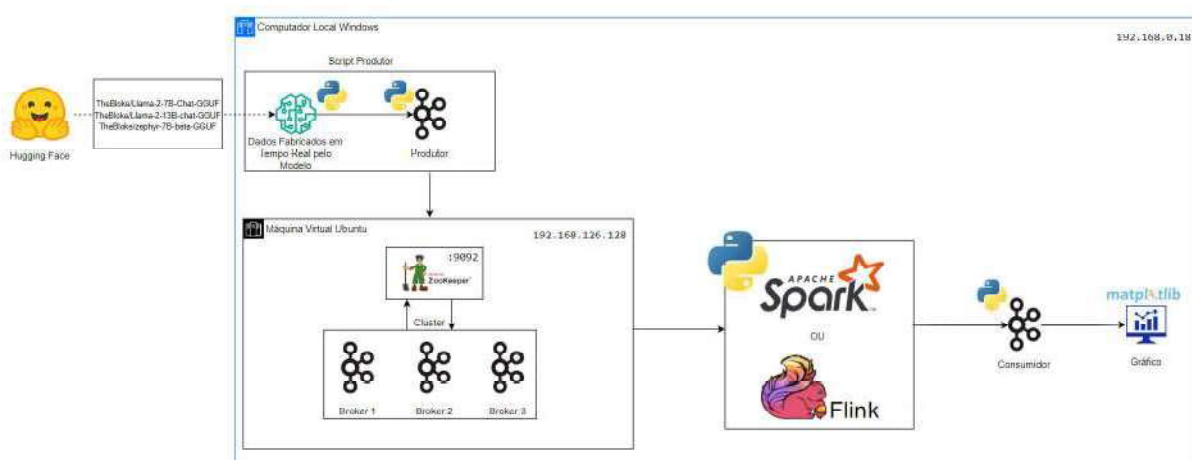
Figura 4 - Bancada Proposta com *Script* Gerando Dados



Fonte: criação do autor³

³ criação do autor utilizando a ferramenta draw.io a partir de imagens dos sites Wikipédia, Kafka Apache, Apache Spark, Apache Flink, Python e Matplotlib. Disponíveis em: <https://en.wikipedia.org/wiki/Apache_ZooKeeper>, <<https://kafka.apache.org/documentation/#zk>>, <<https://spark.apache.org/>>, <<https://flink.apache.org/>>, <<https://www.python.org/>> e <<https://matplotlib.org/>>. Acesso em 26 dez. 2023.

Figura 5 - Bancada Proposta com Grandes Modelos de Linguagem Gerando Dados



Fonte: criação do autor⁴

A diferença entre as bancadas da Figura 4 e Figura 5 é apenas a adição dos Grandes Modelos de Linguagem como geradores dos dados, que são palavras, que foram agrupadas e contabilizadas. Enquanto na bancada da Figura 4, os dados são gerados de acordo com 3 palavras estáticas ou 20 palavras estáticas escolhidas de forma aleatória, no cenário da bancada da Figura 5, por haver mais aleatoriedade devido ao uso desses modelos, a quantidade de colunas, correspondente à quantidade de palavras contabilizadas é imprevisível e pode variar desde algumas unidades até várias dezenas ou até mais.

Para as abordagens da bancada da Figura 4 (3 palavras estáticas e 20 palavras estáticas, escolhidas de forma aleatória), a atuação é semelhante a um cenário de compras de varejo para três produtos ou de compras para vinte produtos.

Já para as abordagens da bancada da Figura 5 (três modelos distintos gerando comentários sobre produtos), o cenário se assemelha ao recebimento de comentários de *feedback* em um site sobre produtos adquiridos, onde as palavras são separadas de forma a se observar quais são os termos mais comentados. Como os Grandes Modelos de Linguagem precisam de instruções em linguagem natural,

⁴ criação do autor utilizando a ferramenta draw.io a partir de imagens dos sites Wikipédia, Kafka Apache, Apache Spark, Apache Flink, Python, Matplotlib e Hugging Face. Disponíveis em: <https://en.wikipedia.org/wiki/Apache_ZooKeeper>, <<https://kafka.apache.org/documentation/#zk>>, <<https://spark.apache.org/>>, <<https://flink.apache.org/>>, <<https://www.python.org/>>, <<https://matplotlib.org/>> e <<https://huggingface.co/>>. Acesso em 26 dez. 2023.

foi elaborado um *prompt*, que fornece essa instrução juntamente com alguns exemplos de comentários sobre esses produtos, conforme Figura 6.

Figura 6 - Prompt dos Grandes Modelos de Linguagem

```
#Iterating through words using prompt to create comments
for i in range(50):
    output = llm("""Crie um comentario bastante curto positivo ou negativo sobre um produto que voce comprou recentemente. Use os exemplos para melhorar os resultados.
    Comentário: Eu odeio meu liquidificador.
    Comentário: Eu amo meu microondas.
    Comentário: Meu computador é incrível.
    Comentário: Meu mouse de computador não funciona muito bom.
    Comentário: Helócio muito bom. Bem melhor que o meu antigo.
    Comentário: O fogão que comprei já está com defeito. Não recomendo não.
    Comentário: Melhor ar condicionado que já comprei. Funciona perfeitamente.
    Comentário: Meu celular não funciona direito. Me arrependo de ter comprado.
    Comentário: Comprei esse modelo de fone de ouvido ontem. Hoje eu abri e veio com vários defeitos.
    Comentário: Paguei muito caro desse freezer e não valeu a pena.
    Comentário: Muito bom e barato esse purificador de água.
    Comentário: Ventilador perfeito. Gostei bastante.
    Comentário: Achei terrível esse roteador. Muito barulhento e difícil de configurar.
    Comentário: Achei que esse lustre seria melhor. Não gostei.
    Comentário: """, max_tokens=30, stop=["Comentário:"], stream=False, repeat_penalty=1.15)
```

Fonte: criada pelo autor

Além disso, foram configurados o número máximo de tokens para 30, um conjunto de caracteres que determina a parada (o termo em questão foi “Comentário:”) e uma penalidade de repetição de 1.15.

O modo de *stream* de geração dos modelos também foi desativado para que fosse possível utilizar uma divisão de palavras de acordo com espaços, já que os modelos retornam *tokens*, que são trechos de caracteres (RAF, [s.d.]). Essa divisão de palavras foi possível através da função *split* presente no Python (“Python String split() Method”, [s.d.]).

Outras configurações necessárias para o bom funcionamento do fluxo de dados foram o uso de uma configuração segundo o “How Could I Stop Printing Time Execution · Issue #999 · ggerganov/llama.cpp” (2023), para evitar resultados no console constantemente, o uso da configuração “earliest” no *auto_reset_offset*, segundo POWERS; ARTHUR ([s.d.]), para evitar perdas de mensagens, o uso de modelos do tipo GGUF, para evitar problemas na execução, conforme registrado em “AssertionError When Using llama-cpp-python in Google Colab” (2023), e, por fim, a remoção de pontuações, que não apresentam relevância no cenário em questão e foram retiradas de acordo com MANJEET_04 (2023b).

Para o uso da visualização de dados, foram usados também recursos da biblioteca Matplotlib que melhoram o desempenho e a interação gráfica, conforme descrito em

“Performance — Matplotlib 3.8.2 Documentation” ([s.d.]), SRINIVASA RAO POLADI (2018) e “Working in Interactive Mode” ([s.d.]).

Uma deleção de arquivos de *checkpoint* ou outros que pudessem impactar no desempenho das tecnologias feita de forma constante também permitiu que o erro descrito em SUMUKH (2019) não ocorresse.

Segundo um estudo realizado pela Neotrust, a quantidade de compras online em 2021 (no período da pandemia) foi de 78,5 milhões no Brasil (REDAÇÃO, 2021). A partir disso, foi possível estabelecer um valor como premissa de compras para simular um monitoramento de compras online de forma a aproximar a fabricação dos dados da realidade, restringindo as compras a produtos específicos (primeiramente com 3 produtos e depois com 20), ao mesmo tempo que mantém a quantidade dessas compras. O racional utilizado para essa simulação de compras foi então que, com 78,5 milhões de compras ao ano, haveria aproximadamente 2,49 compras por segundo, ou, então, aproximando, 1 compra a cada 0,5 segundos. Usando esse número como base das compras, foi configurada uma espera de 0,5 segundos nos *scripts* de forma a não haver um processamento muito rápido, com exceção daqueles casos em que os Grandes Modelos de Linguagem estavam sendo utilizados, já que naturalmente possuem um tempo de execução maior.

Para duas abordagens utilizadas (3 e 20 palavras estáticas escolhidas de forma aleatória), foi então usada uma amostra de 500 (quinhentos) exemplos de mensagens de dados, sendo cada uma enviada a cada 0,5 segundos de forma a simular compras online (novamente, restringindo e ao mesmo tempo ampliando o cenário de três e vinte produtos para as opções totais de consumidores online).

Para a terceira abordagem, foram utilizados os Grandes Modelos de Linguagem para gerar palavras aleatórias simulando palavras em comentários de produtos. O intervalo de palavras coletadas depende exclusivamente do processamento da máquina e do modelo em questão, não sendo necessário configurar um tempo de espera, já que esses modelos têm uma velocidade de resposta maior do que as palavras estáticas escolhidas de forma aleatória. Essas frases são então fracionadas por palavras, que são contabilizadas graficamente. Para uma coletânea relevante, foram geradas 50 iterações de comentários, com diferentes quantidades de palavras

em cada iteração, já que essa é uma característica desses tipos de modelos generativos.

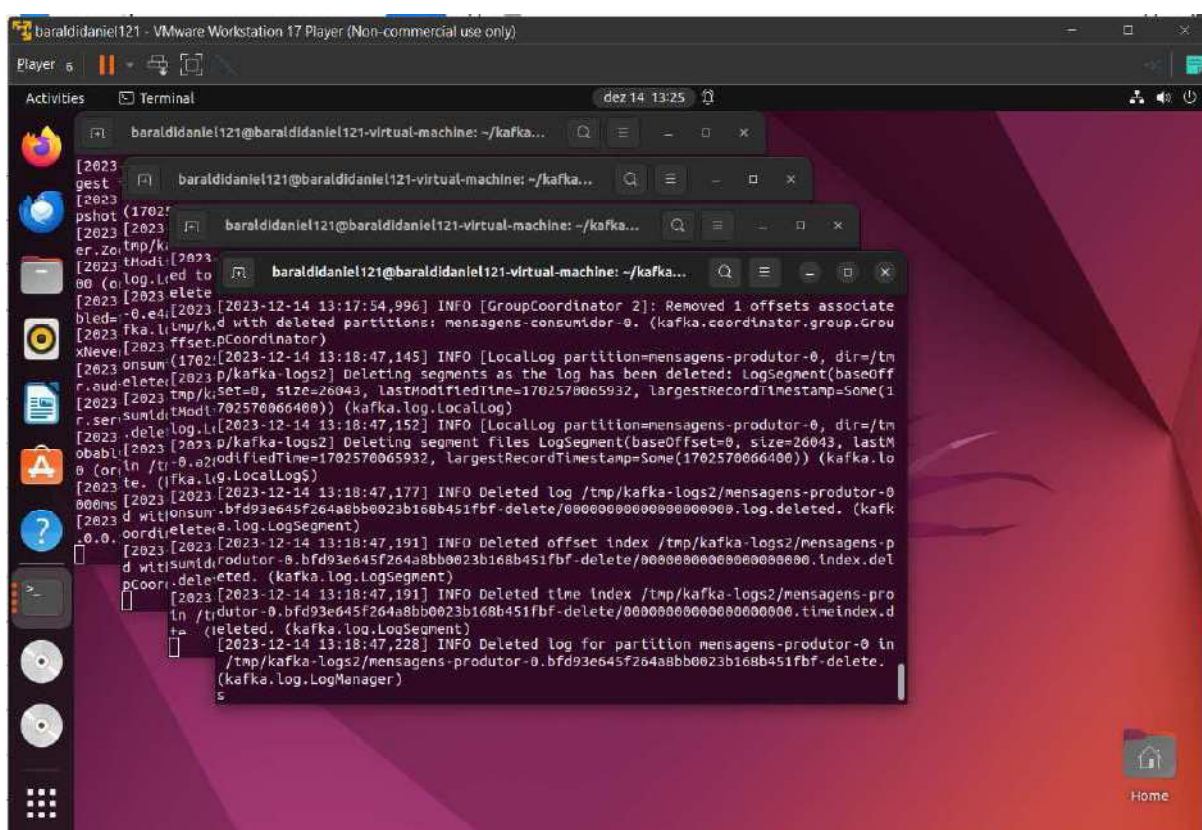
Os *scripts* executados seguem a ordem de execução como *Script* Consumidor, *Script* PySpark ou PyFlink e *Script* Produtor ou então *Script* Consumidor, *Script* Produtor e *Script* PyFlink, já que algumas vezes os Grandes Modelos de Linguagem levam um tempo para iniciarem e isso pode resultar em um erro na execução por não haver mensagens chegando em um primeiro momento para o *script* do PyFlink. Esse foi um desafio da bancada por ser realizado de forma manual e precisar ser executado no tempo correto.

Por fim, os registros de tempo e consumo computacional, além das palavras e frases em questão são registradas em arquivos do tipo txt, para posterior análise e conferência utilizando o Google Sheets para gerar as tabelas de métricas como média e desvio padrão. Assim, o uso do método *flush* durante essa escrita foi bastante relevante e foram seguidas instruções conforme descrito em RITURAJSAHA (2022).

4 RESULTADOS

Após o cluster preparado (Figura 7), os três *scripts* foram executados na máquina Windows. Para cada vez que os *scripts* foram executados, foi realizada uma deleção de tópicos e também de demais use documents que poderiam influenciar no resultado das tecnologias. Possíveis *checkpoints* das tecnologias foram apagados também antes dos *scripts* serem executados.

Figura 7 - Cluster com Zookeeper e Três Brokers para Apache Kafka no VMware



Fonte: criada pelo autor a partir do VMware Workstation Player 17

O último *script*, referente ao consumidor, faz com que um gráfico do Matplotlib seja aberto e então ele é exibido vazio por alguns segundos antes dos dados serem exibidos.

Os resultados deste trabalho foram realizados a partir da coleta de dados enquanto os *scripts* eram executados. Assim, os gráficos foram obtidos durante a execução dos *scripts* do fluxo de dados e os resultados das métricas de tempo, CPU e RAM

foram coletados durante a execução em laço de repetição da leitura das mensagens dos tópicos do Apache Kafka em um tópico específico para leitura.

Após a coleta dos dados de amostra, as operações necessárias foram realizadas com base nos valores obtidos e um comparativo final foi realizado.

De acordo com os objetivos deste trabalho, foram obtidos Resultado da Visualização, Resultado do Tempo de Atualização, Resultado do Consumo de Recurso Computacional, Resultado dos Grandes Modelos de Linguagem e Resultado Geral.

Os gráficos dos resultados em questão seguem a seguinte ordem, dividida entre os tipos 1 e 2, de acordo com as tecnologias utilizadas.

- Resultado da Visualização
 - Sem tratamento gráfico
 - Tipo 1 (Apache Spark/*micro-batches*):
 - Geração de 3 Colunas/Palavras
 - Geração de 20 Colunas/Palavras
 - Modelo de Linguagem Grande (llama 7b)
 - Modelo de Linguagem Grande (llama 13b)
 - Modelo de Linguagem Grande (zephyr 7b)
 - Tipo 2 (Apache Flink/*streaming*):
 - Geração de 3 Colunas/Palavras
 - Geração de 20 Colunas/Palavras
 - Modelo de Linguagem Grande (llama 7b)
 - Modelo de Linguagem Grande (llama 13b)
 - Modelo de Linguagem Grande (zephyr 7b)
 - Com tratamento gráfico
 - Tipo 1 (Apache Spark/*micro-batches*):
 - Geração de 3 Colunas/Palavras
 - Geração de 20 Colunas/Palavras
 - Modelo de Linguagem Grande (llama 7b)
 - Modelo de Linguagem Grande (llama 13b)
 - Modelo de Linguagem Grande (zephyr 7b)

- Tipo 2 (Apache Flink/*streaming*):
 - Geração de 3 Colunas/Palavras
 - Geração de 20 Colunas/Palavras
 - Modelo de Linguagem Grande (llama 7b)
 - Modelo de Linguagem Grande (llama 13b)
 - Modelo de Linguagem Grande (zephyr 7b)
- Resultado do Tempo de Atualização
 - Tipo 1 (Apache Spark/*micro-batches*):
 - Geração de 3 Colunas/Palavras
 - Geração de 20 Colunas/Palavras
 - Modelo de Linguagem Grande (llama 7b)
 - Modelo de Linguagem Grande (llama 13b)
 - Modelo de Linguagem Grande (zephyr 7b)
 - Tipo 2 (Apache Flink/*streaming*):
 - Geração de 3 Colunas/Palavras
 - Geração de 20 Colunas/Palavras
 - Modelo de Linguagem Grande (llama 7b)
 - Modelo de Linguagem Grande (llama 13b)
 - Modelo de Linguagem Grande (zephyr 7b)
- Resultado do Consumo de Recursos Computacionais
 - Tipo 1 (Apache Spark/*micro-batches*):
 - Geração de 3 Colunas/Palavras
 - Geração de 20 Colunas/Palavras
 - Modelo de Linguagem Grande (llama 7b)
 - Modelo de Linguagem Grande (llama 13b)
 - Modelo de Linguagem Grande (zephyr 7b)
 - Tipo 2 (Apache Flink/*streaming*):
 - Geração de 3 Colunas/Palavras
 - Geração de 20 Colunas/Palavras
 - Modelo de Linguagem Grande (llama 7b)
 - Modelo de Linguagem Grande (llama 13b)
 - Modelo de Linguagem Grande (zephyr 7b)

4.1 RESULTADO DA VISUALIZAÇÃO

4.1.1 SEM TRATAMENTO

Para definir boas práticas de visualização, foi necessário executar os *scripts* primeiramente sem nenhum tipo de tratamento gráfico. Assim, o único recurso adaptativo do gráfico foi em relação ao eixo vertical, que é algo nativo, permitindo uma atualização constante sem necessitar de configuração.

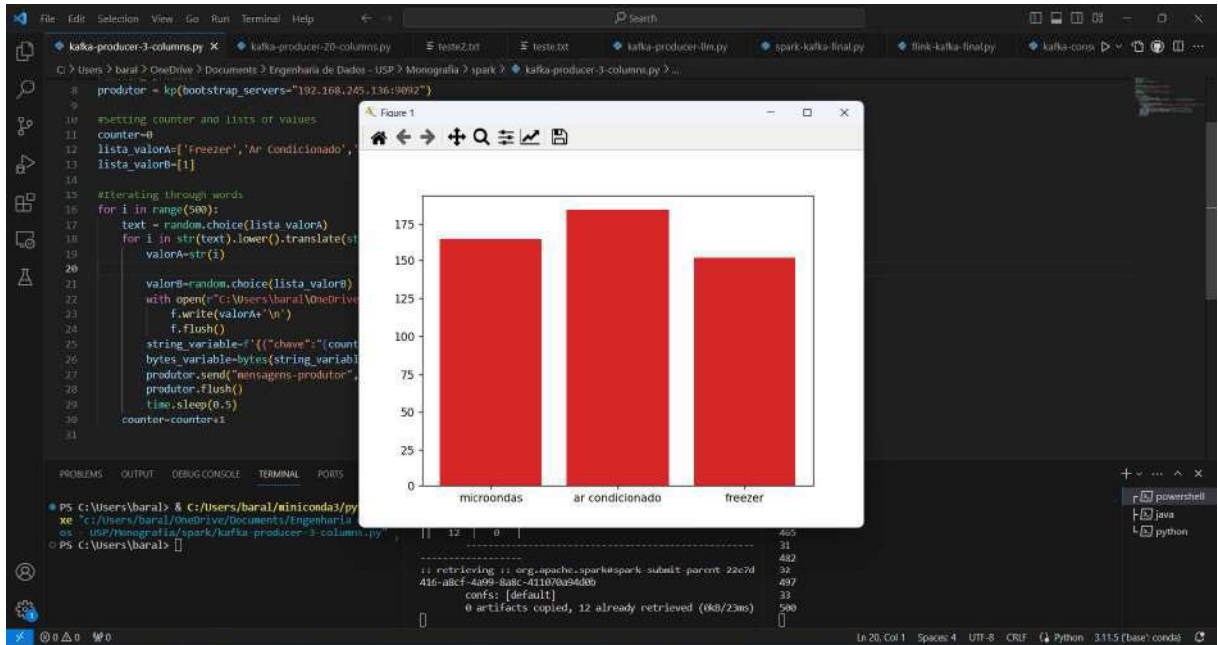
As principais diferenças foram registradas a seguir, com o restante das visualizações e tabelas de resultados completos presentes no Apêndice B.

4.1.1.1 PYSPARK

Foram capturadas figuras de progressão da visualização do PySpark, bem como tabelas indicando as quantidades de dados e métricas, descritas adiante na seção, divididas entre as cinco abordagens propostas (três colunas, vinte colunas, uso do modelo Llama de 7 bilhões de parâmetros, uso do modelo Llama de 13 bilhões de parâmetros e uso do modelo Zephyr de 7 bilhões de parâmetros).

4.1.1.1.1 TRÊS COLUNAS/PRODUTOS E SEM TRATAMENTO (500 EXEMPLOS)

Figura 8 - Visualização Final



Fonte: criada pelo autor

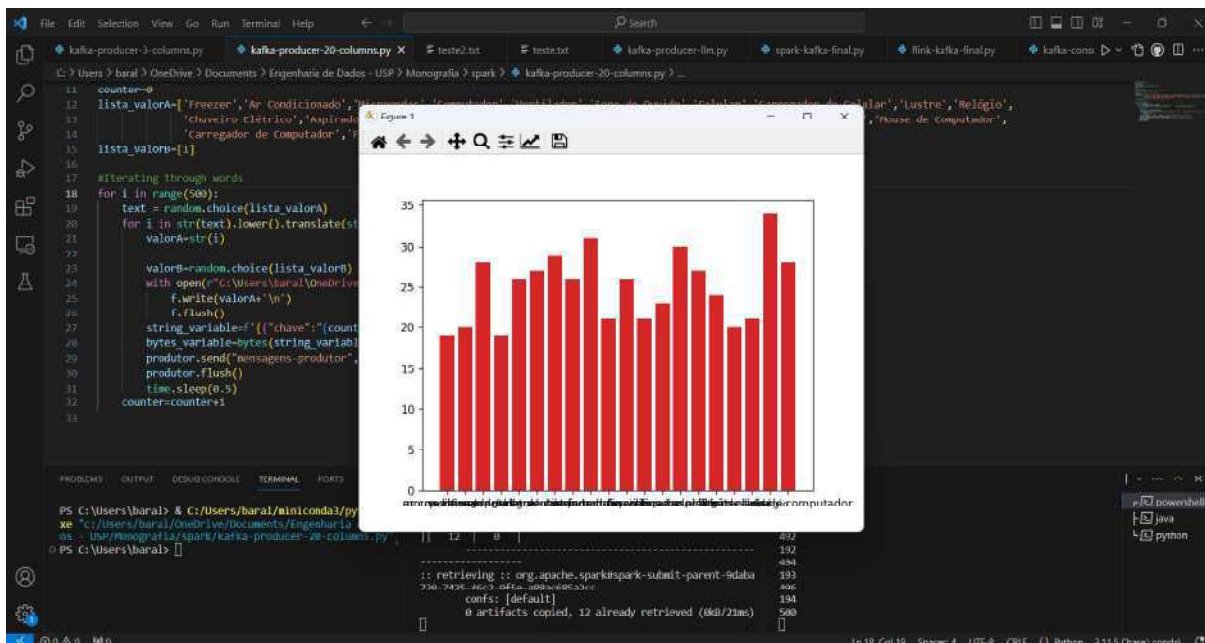
Tabela 3 - Amostras por Palavra

Gráfico de 3 Colunas	Contagem Total de Amostras	Contagem Freezer	Contagem Ar Condicionado	Contagem Microondas
	500	152	184	164

Fonte: elaborada pelo autor

4.1.1.1.2 VINTE COLUNAS/PRODUTOS E SEM TRATAMENTO (500 EXEMPLOS)

Figura 9 - Visualização Final



Fonte: criada pelo autor

Tabela 4 - Contagem Total de Amostras com 500 Exemplos

Contagem Ar Condicionado	Contagem Aspirador	Contagem Carregador de Celular	Contagem Carregador de Computador	Contagem Celular
19	27	19	27	21

Contagem Chuveiro Elétrico	Contagem Computador	Contagem Filtro de Linha	Contagem Fogão	Contagem Fone de Ouvido
30	23	20	24	21

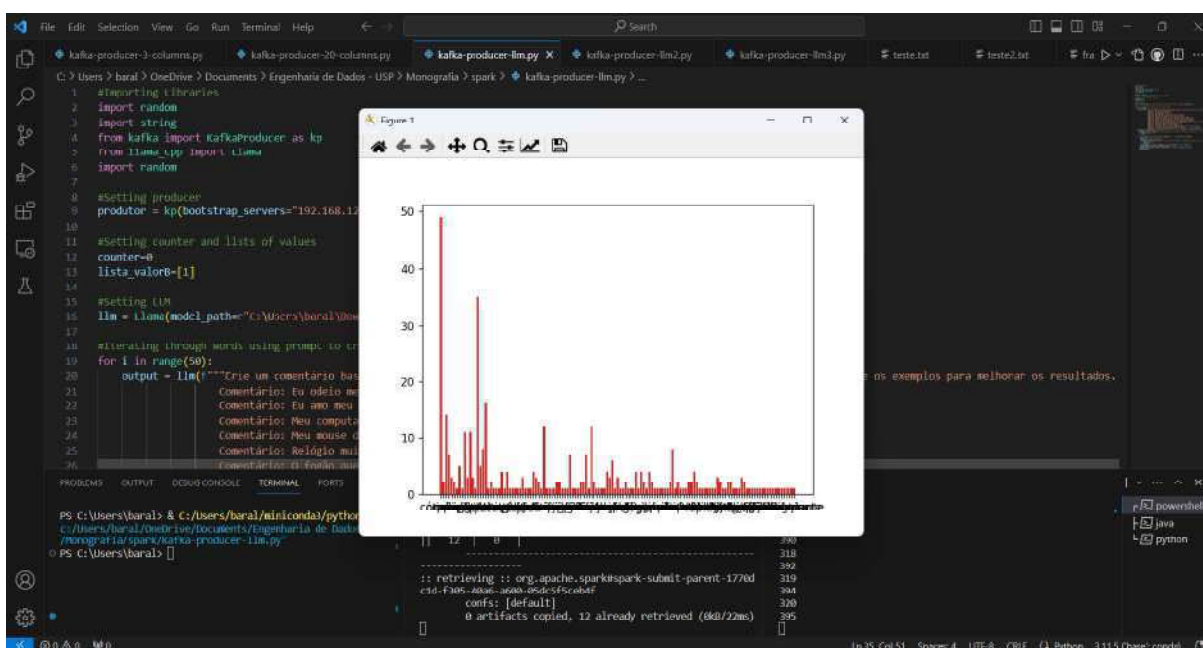
Contagem Freezer	Contagem Liquidificador	Contagem Lustre	Contagem Microondas	Contagem Mouse de Computador
26	21	26	31	20

Contagem Purificador de Água	Contagem Relógio	Contagem Roteador	Contagem Teclado de Computador	Contagem Ventilador
28	34	26	28	29

Fonte: criada pelo autor

4.1.1.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF E SEM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura 10 - Visualização Final



Fonte: criada pelo autor

Tabela 5 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	395	395

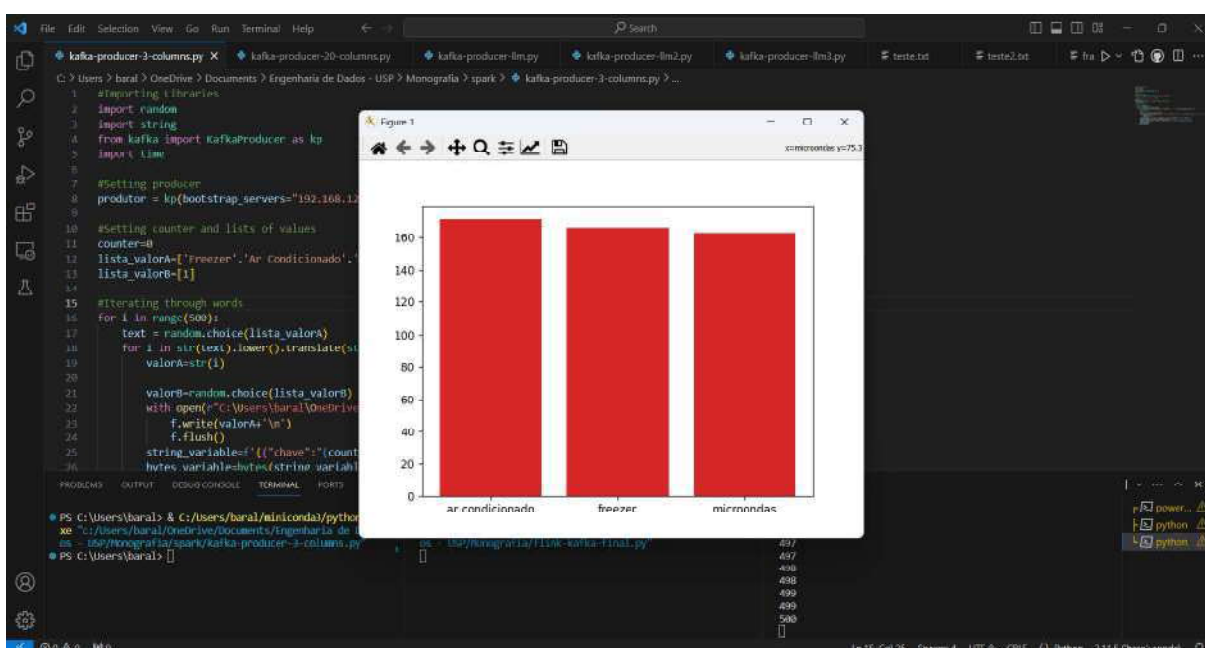
Fonte: criada pelo autor

4.1.1.2 PYFLINK

Nos resultados adiante, também foram demonstradas as mesmas métricas, mas por parte do PyFlink.

4.1.2.1.1 TRÊS COLUNAS/PRODUTOS E SEM TRATAMENTO (500 EXEMPLOS)

Figura 11 - Visualização Final



Fonte: criada pelo autor

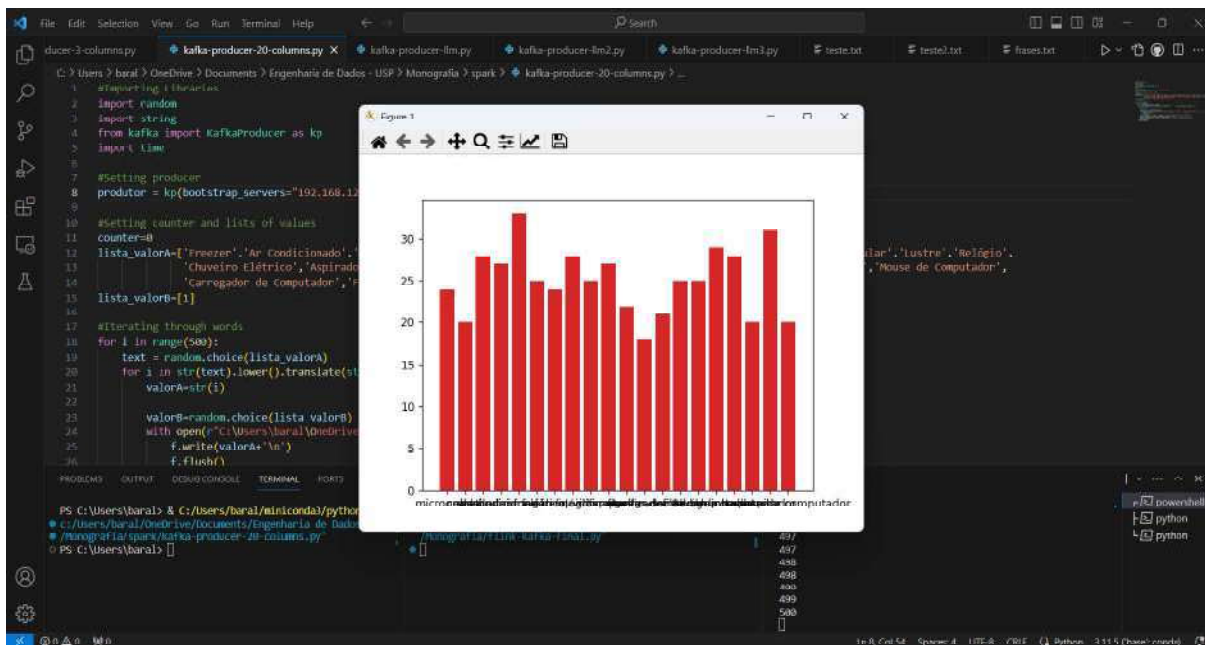
Tabela 6 - Amostras por Palavra

Gráfico de 3 Colunas	Contagem Total de Amostras	Contagem Freezer	Contagem Ar Condicionado	Contagem Microondas
	500	166	171	163

Fonte: criada pelo autor

4.1.2.1.2 VINTE COLUNAS/PRODUTOS SEM TRATAMENTO (500 EXEMPLOS)

Figura 12 - Visualização Final



Fonte: criada pelo autor

Tabela 7 - Contagem Total de Amostras com 500 Exemplos

Contagem Ar Condicionado	Contagem Aspirador	Contagem Carregador de Celular	Contagem Carregador de Computador	Contagem Celular
27	31	18	25	20

Contagem Chuveiro Elétrico	Contagem Computador	Contagem Filtro de Linha	Contagem Fogão	Contagem Fone de Ouvido
33	27	25	25	24

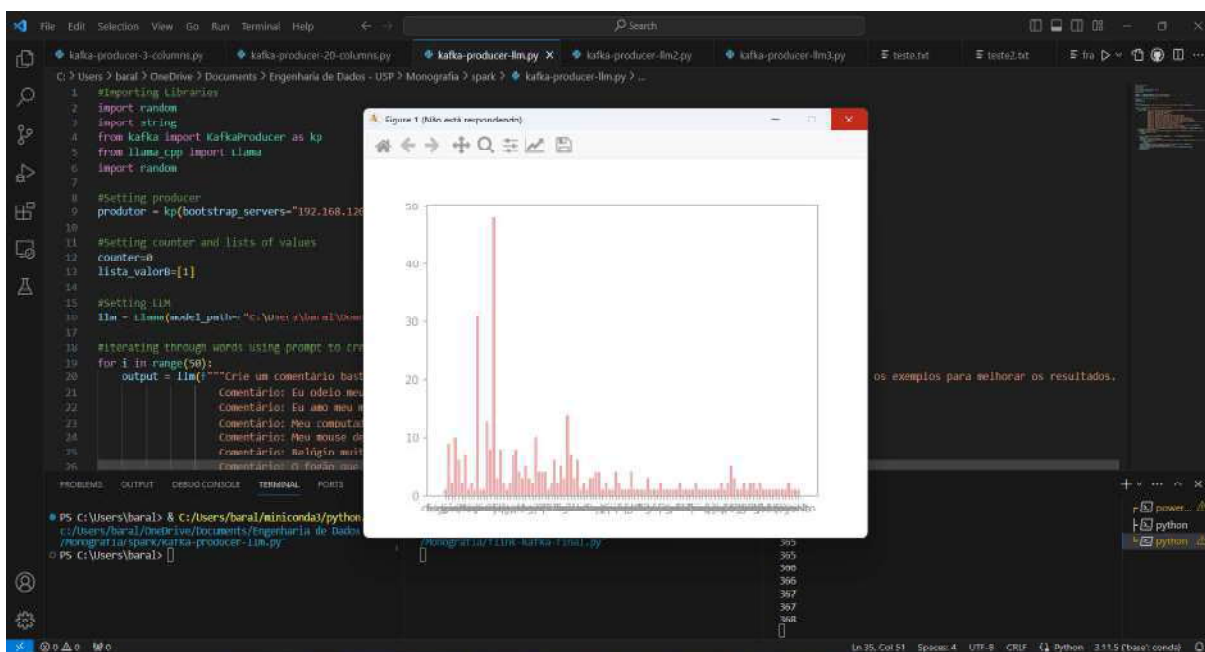
Contagem Freezer	Contagem Liquidificador	Contagem Lustre	Contagem Microondas	Contagem Mouse de Computador
22	25	20	24	20

Contagem Purificador de Água	Contagem Relógio	Contagem Roteador	Contagem Teclado de Computador	Contagem Ventilador
21	28	28	29	28

Fonte: criada pelo autor

4.1.2.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF E SEM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura 13 - Visualização Final



Fonte: criada pelo autor

Tabela 8 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	368	368

Fonte: criada pelo autor

4.1.2 COM TRATAMENTO

Com os resultados obtidos pelos cenários sem tratamento, foram elaborados então tratamentos específicos para endereçar questões referentes à qualidade da visualização e imprevisibilidade dos modelos.

Para o caso do Matplotlib, o eixo vertical se atualiza de forma constante, corrigindo espaçamentos e demais características de escala. Porém, para o eixo horizontal, existe a sobreposição de valores. Além disso, com o uso dos Grandes Modelos de Linguagem, as palavras geradas são muito diversas, fazendo com que a visualização fique comprometida, conforme a Figura 9 e a Figura 10 para o PySpark e a Figura 12 e a Figura 13 para o PyFlink. Com isso, é necessário realizar tratamentos específicos para esses pontos.

Como tratamentos, foram realizados, de acordo com as respectivas documentações como referências:

- Ordenamento dos valores e exibição por maior quantidade (CHRIS, 2022)
- Restrição para exibição apenas dos top 30 valores (MANJEET_04, 2023; “Plotting Top 10 Values in Big Data”, 2022)
- Rotação dos rótulos do eixo horizontal em 90 graus (SCOTTLITTLE, 2016)
- Ajuste de *layout* de acordo com o espaço no gráfico (foi utilizado um recurso nativo da biblioteca matplotlib, `plt.tight_layout`) (BART, 2015)

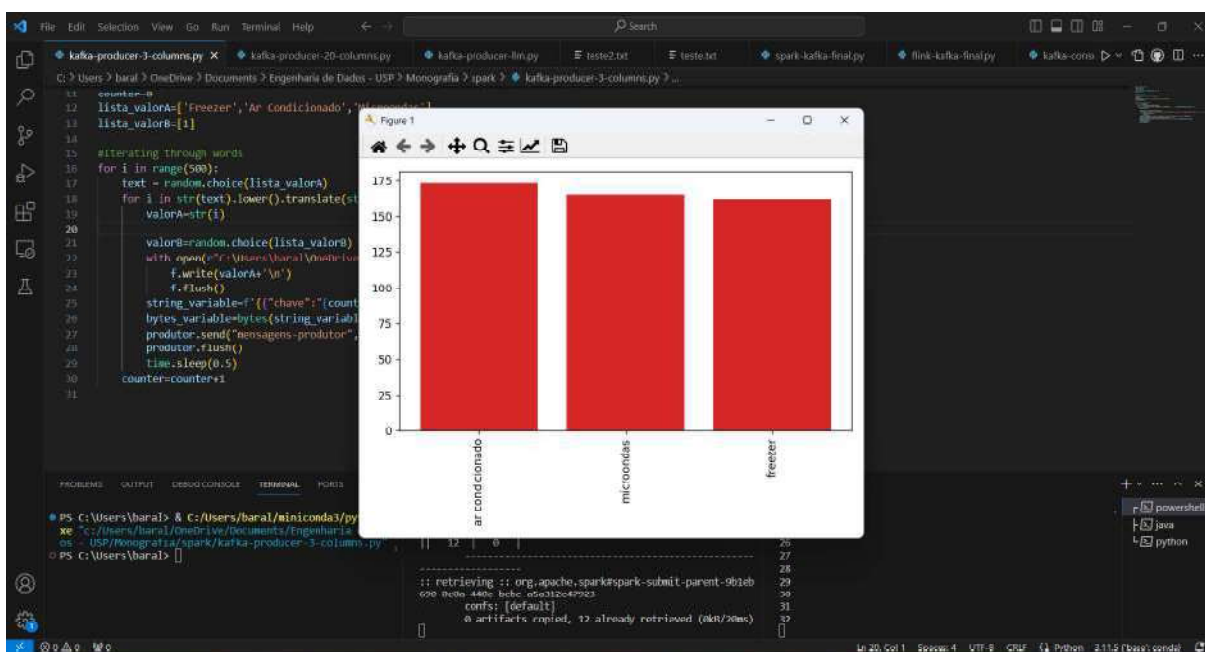
Dessa forma, os gráficos e demais métricas foram, a partir de então, criados com o uso desses tratamentos.

4.1.2.1 PYSPARK

Adiante, foram exibidos os gráficos e as métricas para o PySpark considerando o cenário com o tratamento.

4.1.2.1.1 TRÊS COLUNAS/PRODUTOS E COM TRATAMENTO (500 EXEMPLOS)

Figura 14 - Visualização Final



Fonte: criada pelo autor

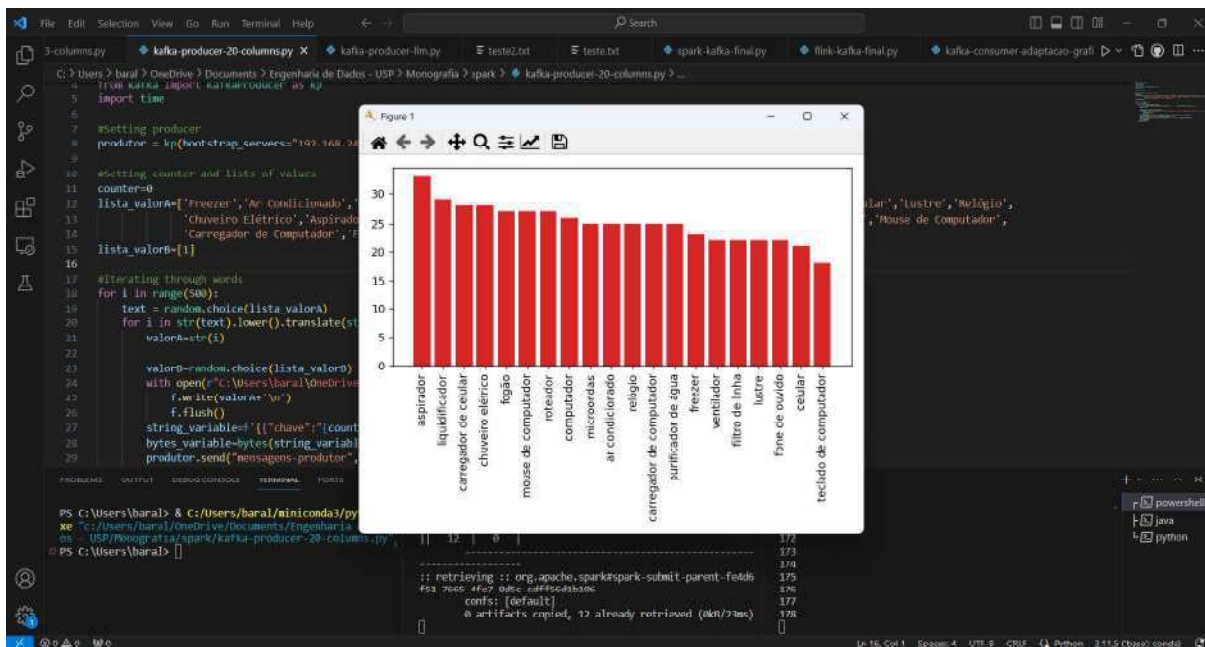
Tabela 9 - Amostras por Palavra

Gráfico de 3 Colunas	Contagem Total de Amostras	Contagem Freezer	Contagem Ar Condicionado	Contagem Microondas
	500	162	173	165

Fonte: criada pelo autor

4.1.2.1.2 VINTE COLUNAS/PRODUTOS E COM TRATAMENTO (500 EXEMPLOS)

Figura 15 - Visualização Final



Fonte: criada pelo autor

Tabela 10 - Contagem Total de Amostras com 500 Exemplos

Contagem Ar Condicionado	Contagem Aspirador	Contagem Carregador de Celular	Contagem Carregador de Computador	Contagem Celular
25	33	28	25	21

Contagem Chuveiro Elétrico	Contagem Computador	Contagem Filtro de Linha	Contagem Fogão	Contagem Fone de Ouvido
28	26	22	27	22

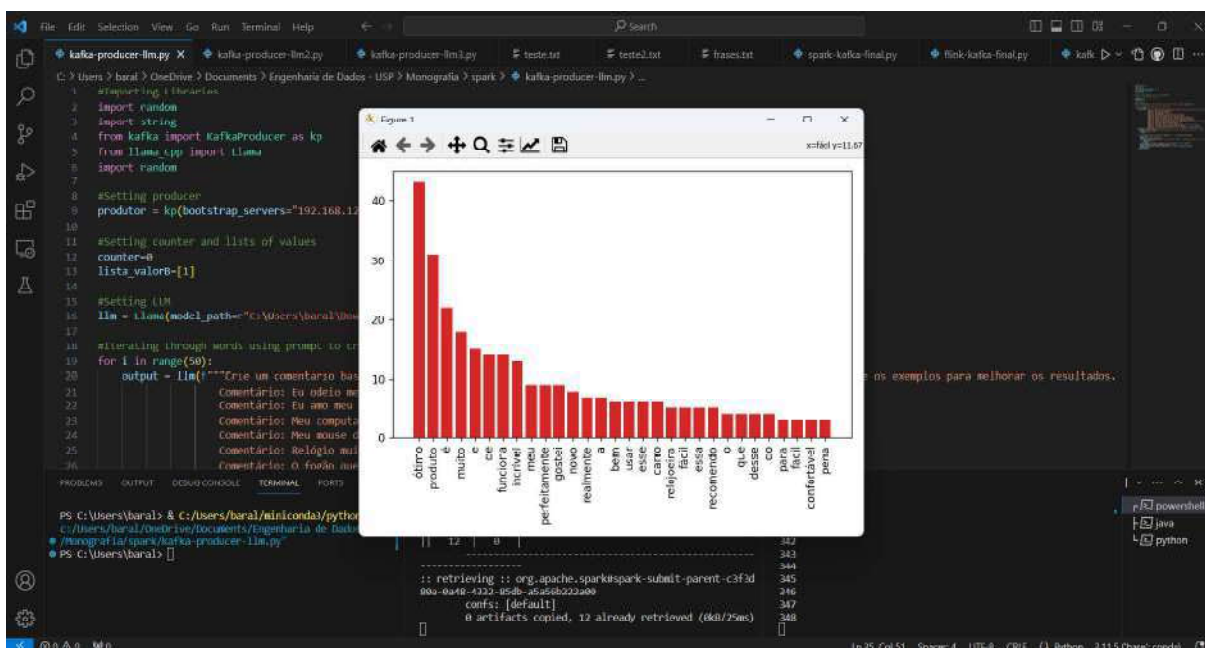
Contagem Freezer	Contagem Liquidificador	Contagem Lustre	Contagem Microondas	Contagem Mouse de Computador
23	29	22	25	27

Contagem Purificador de Água	Contagem Relógio	Contagem Roteador	Contagem Teclado de Computador	Contagem Ventilador
25	25	27	18	22

Fonte: criada pelo autor

4.1.2.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF COM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura 16 - Visualização Final



Fonte: criada pelo autor

Tabela 11 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	406	406

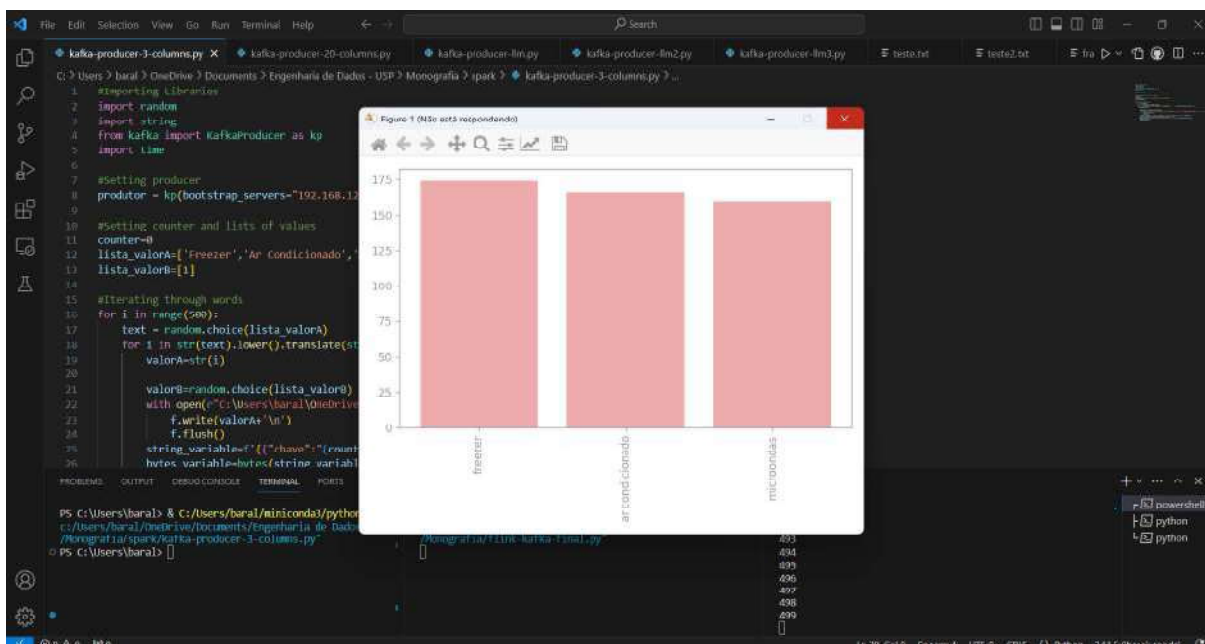
Fonte: criada pelo autor

4.1.2.2 PYFLINK

Da mesma maneira, foram exibidos os gráficos e as métricas adiante para o PyFlink, considerando o cenário com o tratamento.

4.1.2.2.1 TRÊS COLUNAS/PRODUTOS COM TRATAMENTO (500 EXEMPLOS)

Figura 17 - Visualização Final



Fonte: criada pelo autor

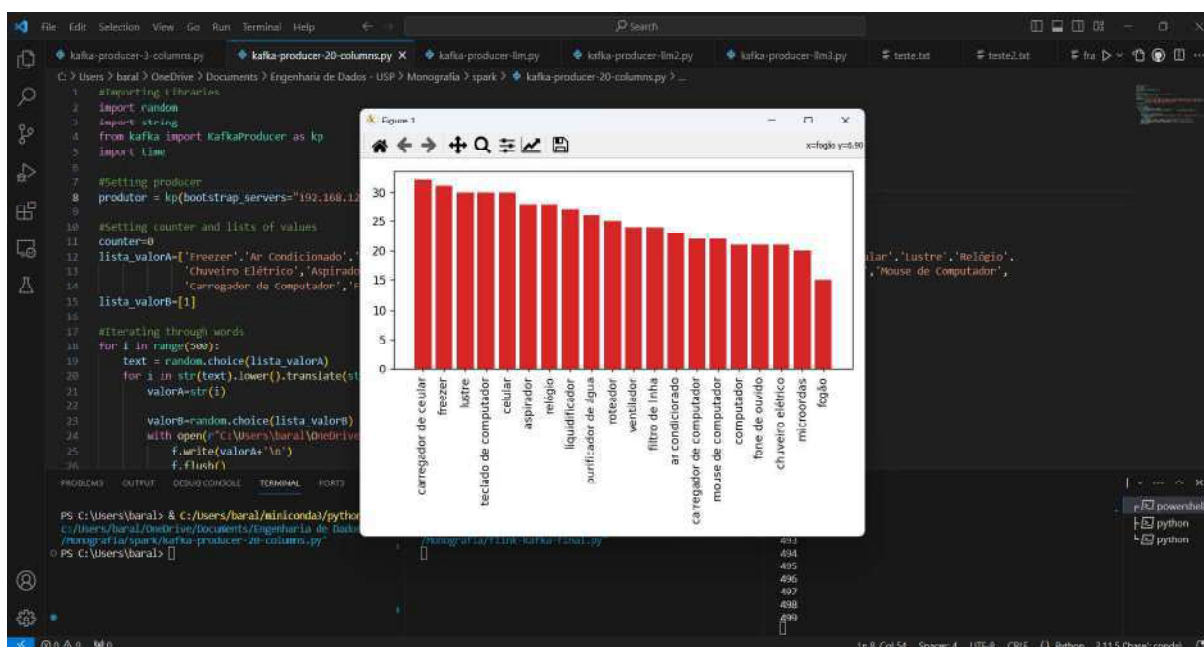
Tabela 12 - Amostras por Palavra

Gráfico de 3 Colunas	Contagem Total de Amostras	Contagem Freezer	Contagem Ar Condicionado	Contagem Microondas
		500	174	166

Fonte: criada pelo autor

4.1.2.2.2 VINTE COLUNAS/PRODUTOS E COM TRATAMENTO (500 EXEMPLOS)

Figura 18 - Visualização Final



Fonte: criada pelo autor

Tabela 13 - Contagem Total de Amostras com 500 Exemplos

Contagem Ar Condicionado	Contagem Aspirador	Contagem Carregador de Celular	Contagem Carregador de Computador	Contagem Celular
23	28	32	22	30

Contagem Chuveiro Elétrico	Contagem Computador	Contagem Filtro de Linha	Contagem Fogão	Contagem Fone de Ouvido
21	21	24	15	21

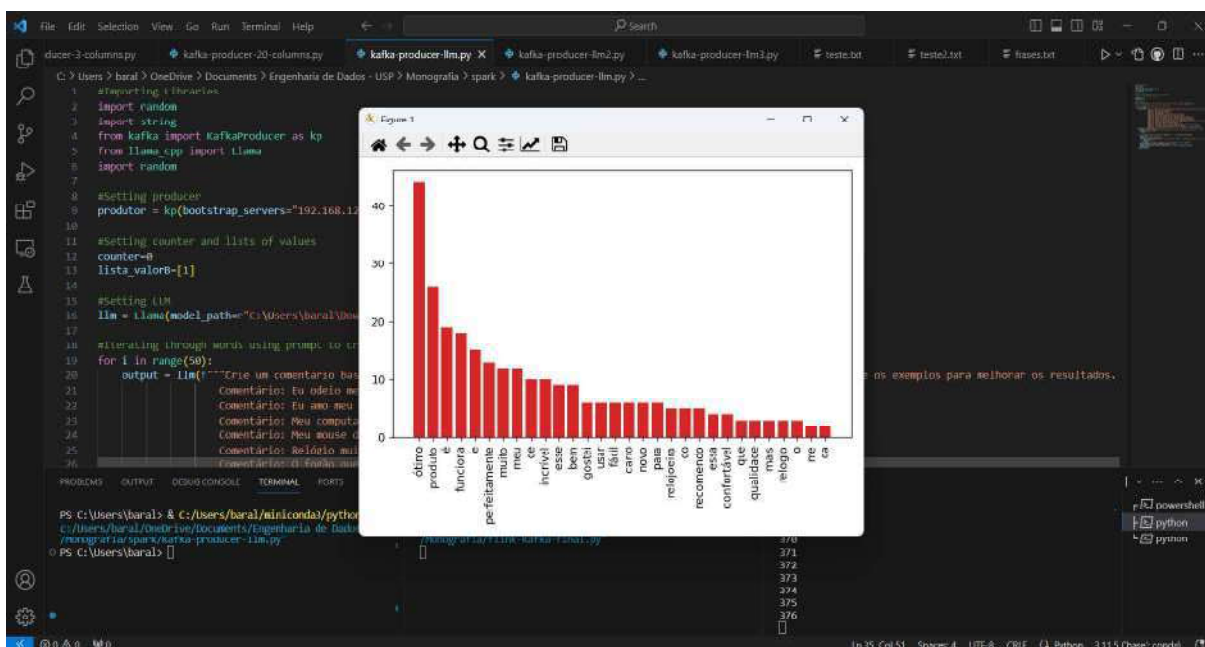
Contagem Freezer	Contagem Liquidificador	Contagem Lustre	Contagem Microondas	Contagem Mouse de Computador
31	27	30	20	22

Contagem Purificador de Água	Contagem Relógio	Contagem Roteador	Contagem Teclado de Computador	Contagem Ventilador
26	28	25	30	24

Fonte: criada pelo autor

4.1.2.2.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF E COM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura 19 - Visualização Final



Fonte: criada pelo autor

Tabela 14 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	377	377

Fonte: criada pelo autor

4.1.2 ANÁLISE DA VISUALIZAÇÃO

As análises dos resultados foram feitas a partir do Google Sheets, com diversos tratamentos nos registros e dados coletados, mas mantendo a integridade e o conteúdo. Esses tratamentos se restringiram ao uso de funções para extração de trechos de texto específicos (JASAITIS, 2022), linhas em branco (FROLOV, 2023) e

extração dos valores de máximo para cada palavra, de forma a saber qual o valor final contabilizado de cada uma dessas palavras (ZACH, 2023).

Os registros de contagens de palavras, como na Tabela 10 e Tabela 11 para o PySpark e Tabela 13 e Tabela 14 para o PyFlink, foram usados como referências de forma a garantir que as mensagens enviadas foram recebidas no final do fluxo de dados.

A partir das progressões gráficas e dos registros em cada laço de repetição, foi possível perceber que as duas tecnologias respondem relativamente bem a mudanças, embora haja diferenças entre as abordagens de *micro-batches* e *streaming*.

A primeira diferença se deu em relação à percepção gráfica de contabilização nas colunas. Enquanto o PySpark (*micro-batches*) apresentou uma mudança mais brusca, o PyFlink (*streaming*) obteve transições constantes e mais suaves. Isso é provavelmente um resultado da característica do PySpark ainda usar um processamento em *batches* simulando *streaming*, mesmo que sejam processamentos constantes.

A segunda diferença se deu em relação às atualizações dos registros. Foi possível perceber que os registros do PyFlink seguiram de forma relacionada (Figura 20), com o mesmo número de linhas de registro e palavras, enquanto o PySpark apresenta certas lacunas de atualização dos registros, focando apenas no valor mais atual, conforme a Figura 21, em que a palavra “ótimo” tem um salto de 1 para 3. Isso contribui para a sensação mais fluida do PyFlink.

Figura 20 - Tratamento e Conferência dos Dados no PyFlink

	C	D	E	F	G	H	I	J	K	L
1	Contagem Database	Palavras Brutas	Palavras_Tratad	Palavra Final	Valor	Valor Tratado	Max Valor			
2	{valorA:"confiável",valorB":1}	{valorA:"confiável",valorB":1}	confiável	confiável	1	1,00	1,00			
3	{valorA:"funciona",valorB":1}	{valorA:"funciona",valorB":1}	funciona	funciona	1	1,00	9,00			
4	{valorA:"frigorífico",valorB":1}	{valorA:"frigorífico",valorB":1}	frigorífico	frigorífico	1	1,00	2,00			
5	{valorA:"para",valorB":1}	{valorA:"para",valorB":1}	para	para	1	1,00	10,00			
6	{valorA:"gostei",valorB":1}	{valorA:"gostei",valorB":1}	gostei	gostei	1	1,00	6,00			
7	{valorA:"este",valorB":1}	{valorA:"este",valorB":1}	este	este	1	1,00	2,00			
8	{valorA:"esse",valorB":1}	{valorA:"esse",valorB":1}	esse	esse	1	1,00	7,00			
9	{valorA:"crianças",valorB":1}	{valorA:"crianças",valorB":1}	crianças	crianças	1	1,00	1,00			
10	{valorA:"desse",valorB":1}	{valorA:"desse",valorB":1}	desse	desse	1	1,00	2,00			
11	{valorA:"adoram",valorB":1}	{valorA:"adoram",valorB":1}	adoram	adoram	1	1,00	1,00			
12	{valorA:"produto",valorB":1}	{valorA:"produto",valorB":1}	produto	produto	1	1,00	31,00			
13	{valorA:"scanner",valorB":1}	{valorA:"scanner",valorB":1}	scanner	scanner	1	1,00	1,00			
14	{valorA:"meus",valorB":1}	{valorA:"meus",valorB":1}	meus	meus	1	1,00	1,00			
15	{valorA:"muito",valorB":1}	{valorA:"muito",valorB":1}	muito	muito	1	1,00	13,00			
16	{valorA:"o",valorB":1}	{valorA:"o",valorB":1}	o	o	1	1,00	0,00			
17	{valorA:"ótimo",valorB":1}	{valorA:"ótimo",valorB":1}	ótimo	ótimo	1	1,00	48,00			
18	{valorA:"frigorífico",valorB":2}	{valorA:"frigorífico",valorB":2}	frigorífico	vale	2	2,00	3,00			
19	{valorA:"vale",valorB":1}	{valorA:"vale",valorB":1}	vale	que	1	1,00	8,00			
20	{valorA:"que",valorB":1}	{valorA:"que",valorB":1}	que	brinquedo	1	1,00	2,00			

Fonte: criada pelo autor

Figura 21 - Tratamento e Conferência dos Dados no PySpark

	A	B	C	D	E	F	G	H	I	J	K	L
1	Contagem Database	Palavras Brutas	Palavras_Tratad	Palavra Final	Valor	Valor Tratado	Max Valor					
2	{valorA:"ótimo",valorB":1.0}	{valorA:"ótimo",valorB":1.0}	ótimo	ótimo	1.0	1,00	49,00					
3	{valorA:"compras",valorB":1.0}	{valorA:"compras",valorB":1.0}	compras	compras	1.0	1,00	2,00					
4	{valorA:"e",valorB":1.0}	{valorA:"e",valorB":1.0}	e	e	1.0	1,00	14,00					
5	{valorA:"para",valorB":1.0}	{valorA:"para",valorB":1.0}	para	para	1.0	1,00	7,00					
6	{valorA:"online",valorB":1.0}	{valorA:"online",valorB":1.0}	online	online	1.0	1,00	3,00					
7	{valorA:"carninho",valorB":1.0}	{valorA:"carninho",valorB":1.0}	carninho	carninho	1.0	1,00	2,00					
8	{valorA:"facil",valorB":1.0}	{valorA:"facil",valorB":1.0}	facil	facil	1.0	1,00	1,00					
9	{valorA:"usar",valorB":2.0}	{valorA:"usar",valorB":2.0}	usar	usar	2.0	2,00	5,00					
10	{valorA:"sapato",valorB":1.0}	{valorA:"sapato",valorB":1.0}	sapato	sapato	1.0	1,00	1,00					
11	{valorA:"e",valorB":1.0}	{valorA:"e",valorB":1.0}	e	e	1.0	1,00	11,00					
12	{valorA:"creme",valorB":1.0}	{valorA:"creme",valorB":1.0}	creme	creme	1.0	1,00	3,00					
13	{valorA:"de",valorB":2.0}	{valorA:"de",valorB":2.0}	de	de	2.0	2,00	11,00					
14	{valorA:"gosto",valorB":1.0}	{valorA:"gosto",valorB":1.0}	gosto	gosto	1.0	1,00	3,00					
15	{valorA:"rápido",valorB":1.0}	{valorA:"rápido",valorB":1.0}	rápido	rápido	1.0	1,00	1,00					
16	{valorA:"produto",valorB":2.0}	{valorA:"produto",valorB":2.0}	produto	produto	2.0	2,00	35,00					
17	{valorA:"gostei",valorB":1.0}	{valorA:"gostei",valorB":1.0}	gostei	gostei	1.0	1,00	5,00					
18	{valorA:"ótimo",valorB":3.0}	{valorA:"ótimo",valorB":3.0}	ótimo	esse	3.0	3,00	8,00					
19	{valorA:"esse",valorB":1.0}	{valorA:"esse",valorB":1.0}	esse	muito	1.0	1,00	16,00					
20	{valorA:"muito",valorB":2.0}	{valorA:"muito",valorB":2.0}	muito	hidratante	2.0	2,00	1,00					

Fonte: criada pelo autor

Outra questão é que esses saltos de atualização do PySpark se intensificaram para apenas três colunas/produtos, devido ao baixo tempo de espera (apenas 0,5 segundo) para a contabilização de uma nova palavra, acabando por serem mais perceptíveis.

Além desse comparativo entre as tecnologias, foi realizado um tratamento na visualização de forma a atingir uma referência de uso de visualização em *streaming*.

Os tratamentos em questão realizados foram, de acordo com as respectivas referências:

- Ordenamento dos valores e exibição por maior quantidade (CHRIS, 2022)
- Restrição para exibição apenas dos top 30 valores (MANJEET_04, 2023; “Plotting Top 10 Values in Big Data”, 2022)
- Rotação dos rótulos do eixo horizontal em 90 graus (SCOTTLITTLE, 2016)
- Ajuste de *layout* de acordo com o espaço no gráfico (foi utilizado um recurso nativo da biblioteca matplotlib, `plt.tight_layout`) (BART, 2015)

Esse tratamento permitiu então que os valores não tivessem mais problemas de sobreposição no eixo horizontal, uma vez que apenas os primeiros 30 maiores valores foram exibidos e também os rótulos de dados foram rotacionados em 90 graus.

Além disso, palavras maiores foram devidamente encaixadas no gráfico através do uso do ajuste de *layout*.

Uma última alteração que contribuiu para que a visualização seja adequada ao cenário de *streaming* foi o uso do ordenamento por maior quantidade, fazendo com que o usuário possa focar nos valores que realmente interessam, são mais significativos e aumentam gradativamente.

Para as demais análises, foi utilizado então esse modelo tido como aceitável para visualização de dados e com os devidos tratamentos para corrigir os problemas causados pelo alto volume e a alta variação de dados desse cenário de *streaming*.

4.2 RESULTADO DO TEMPO DE ATUALIZAÇÃO

As métricas coletadas se referem à adaptação gráfica/tempo de atualização do gráfico. Foi realizada a mudança de 3 colunas/produtos para 20, buscando observar alterações de comportamento das tecnologias e o aumento do tempo de atualização. Além disso, também foi realizada uma mudança usando os Grandes Modelos de Linguagem para avaliar ainda mais mudanças de comportamento referentes a aleatoriedades de palavras e maior volume.

Os tempos de atualização gráfica foram obtidos a partir da referência em RATED_R_SUNDRAM (2023), que permitiu que os devidos registros fossem realizados. Após esses registros, foi calculada uma média e um desvio padrão nos valores de atualização para cada execução de fluxo de dados realizada.

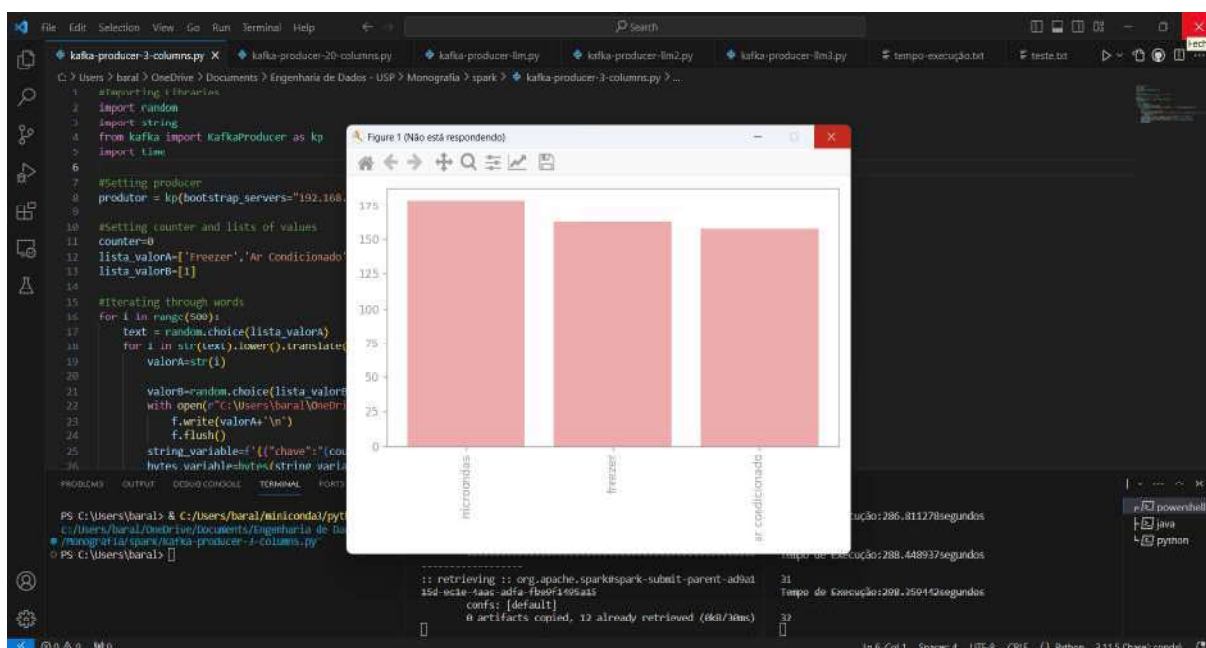
A seguir foram demonstrados os resultados obtidos de algumas visualizações com seus respectivos tempos de atualização gráfica. Os demais resultados se encontram no Apêndice B.

4.2.1.1 PYSPARK

Para o tempo de atualização, foram coletados gráficos de progressão gráfica e métricas de média e desvio padrão desses tempos de atualização, baseando-se de agora em diante apenas no cenário já com o tratamento gráfico e direcionado para o PySpark.

4.2.1.1.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura 22 - Visualização Final



Fonte: criada pelo autor

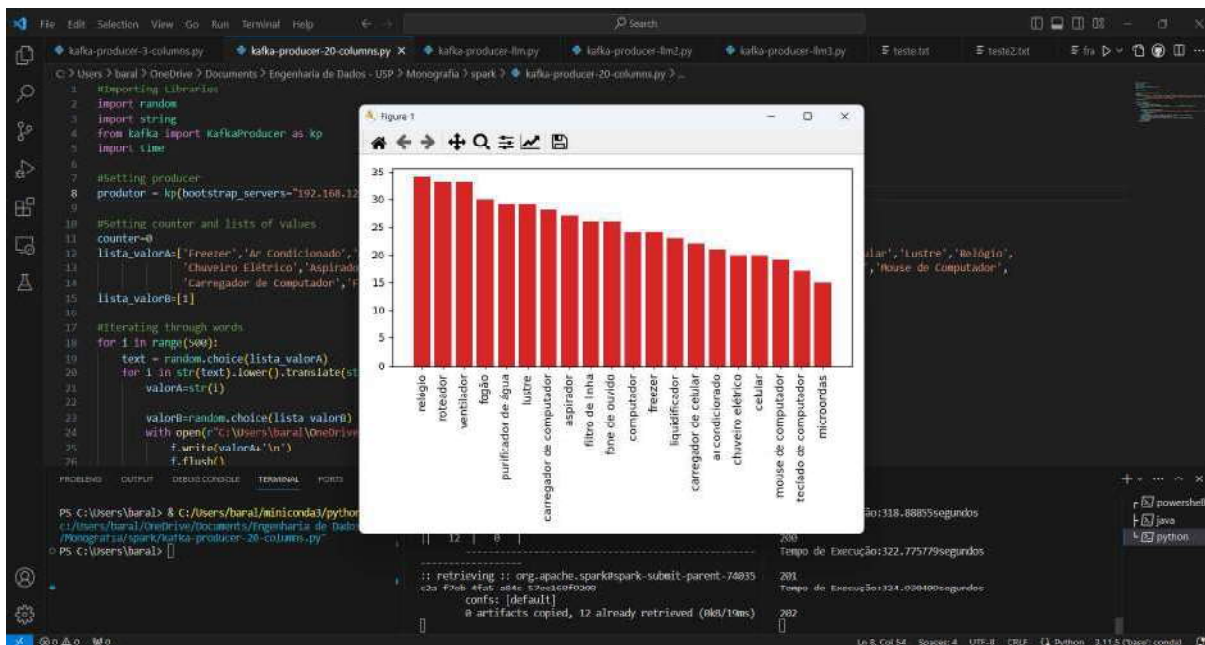
Tabela 15 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
9,041	7,875

Fonte: criada pelo autor

4.2.1.1.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura 23 - Visualização Final



Fonte: criada pelo autor

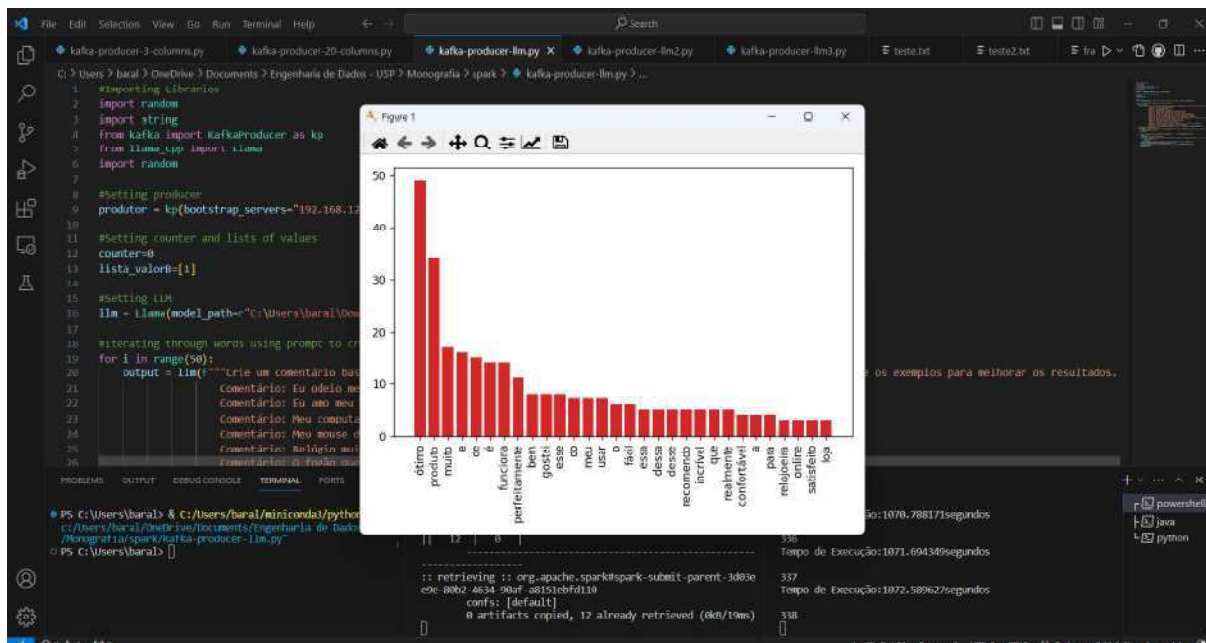
Tabela 16 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
1,601	3,072

Fonte: criada pelo autor

4.2.1.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura 24 - Visualização Final



Fonte: criada pelo autor

Tabela 17 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
3,164	7,459

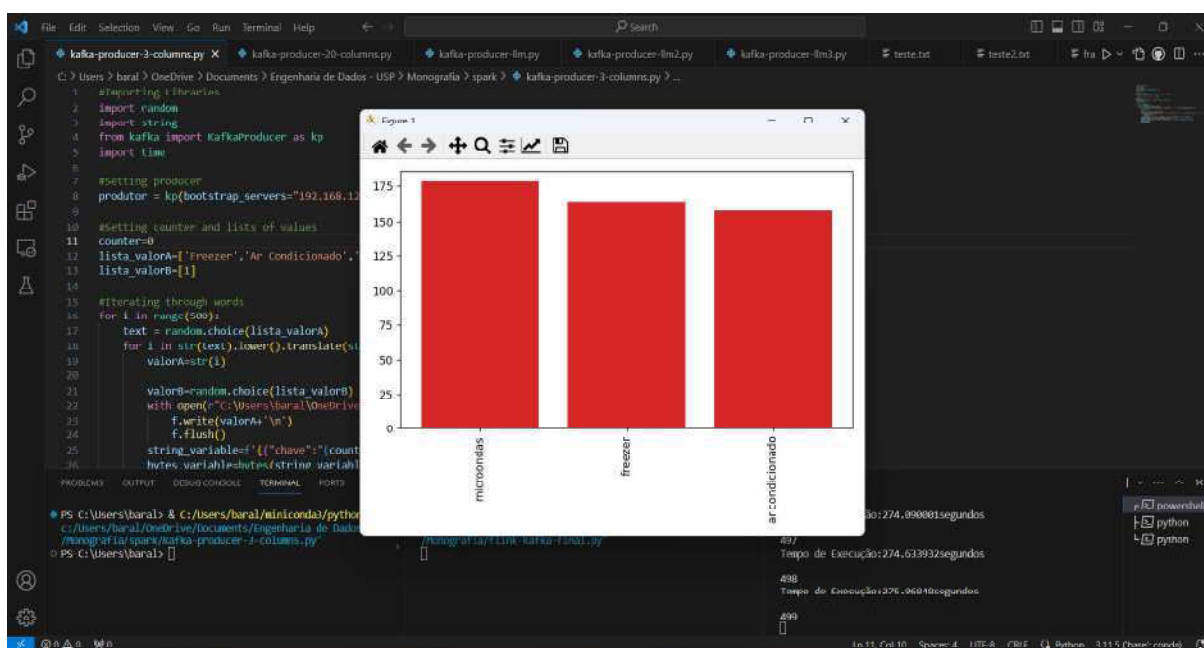
Fonte: criada pelo autor

4.2.1.2 PYFLINK

Igualmente, foram realizadas adiante as coletas para o PyFlink.

4.2.1.2.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura 25 - Visualização Final



Fonte: criada pelo autor

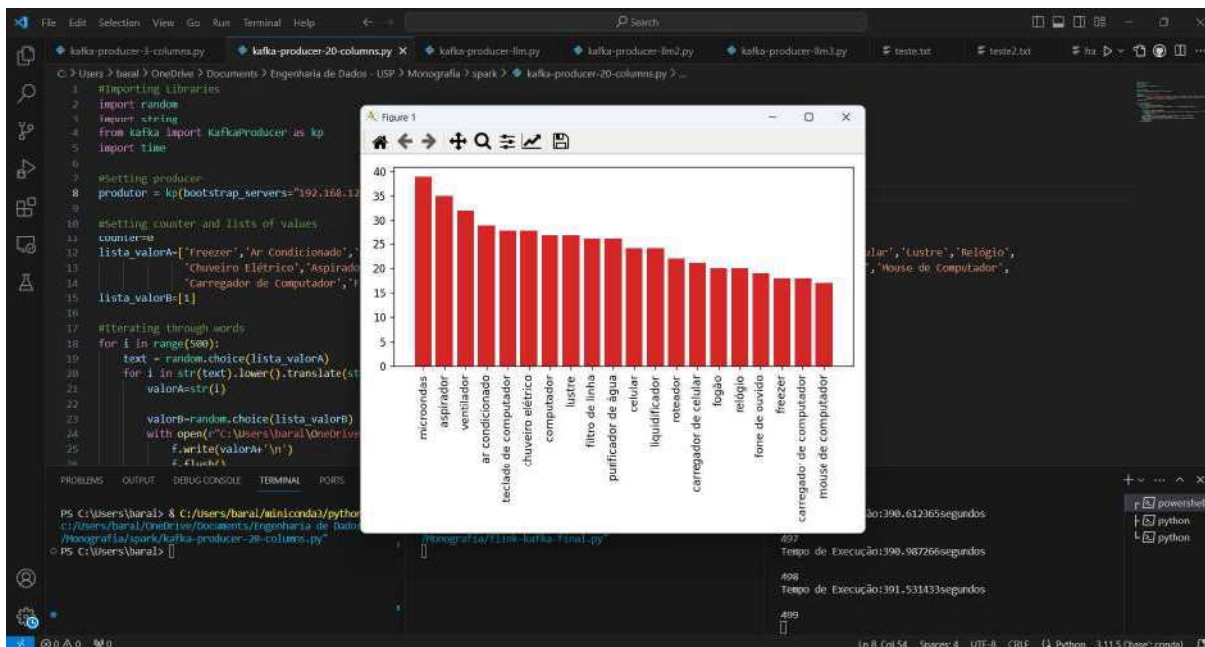
Tabela 18 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
0,550	1,492

Fonte: criada pelo autor

4.2.1.2.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura 26 - Visualização Final



Fonte: criada pelo autor

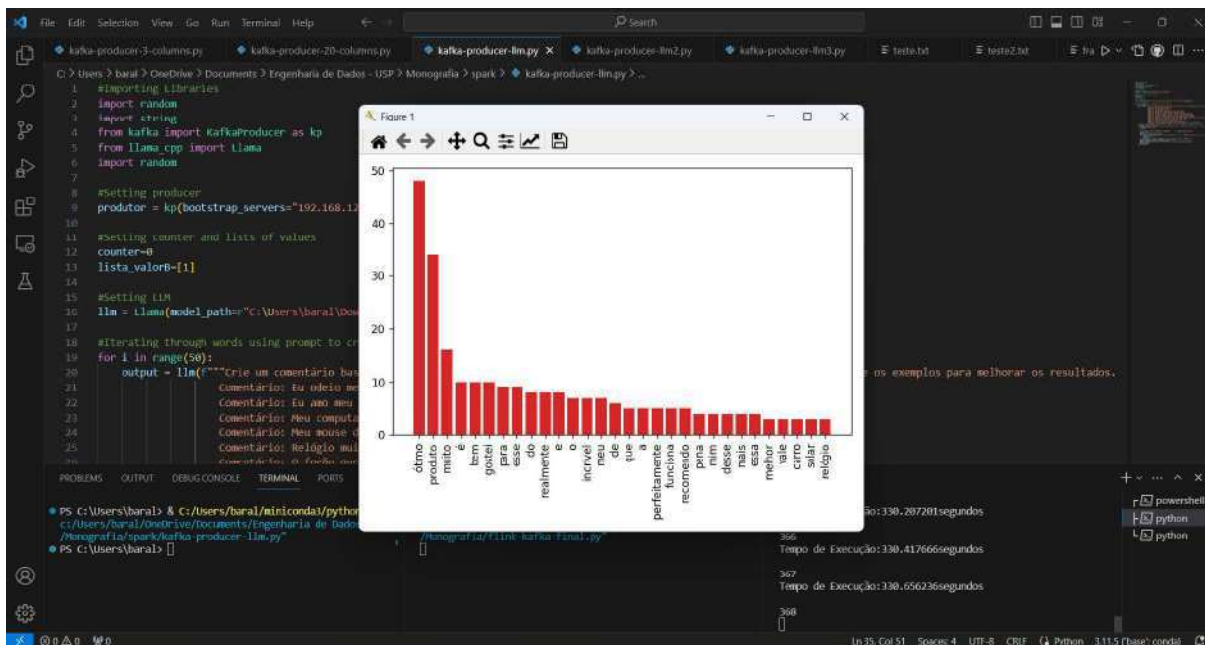
Tabela 19 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
0,783	6,382

Fonte: criada pelo autor

4.2.1.2.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura 27 - Visualização Final



Fonte: criada pelo autor

Tabela 20 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
0,896	4,754

Fonte: criada pelo autor

4.2.1.3 ANÁLISE DO TEMPO DE ATUALIZAÇÃO

A partir das coletas parciais da Figura 24 e Tabela 17 para o PySpark, por exemplo, e Figura 27 e Tabela 20 para o PyFlink, foi realizado o consolidado dos tempos de atualização dos fluxos de dados, que pode ser visualizado na Tabela 21, com suas médias, e, na Tabela 22, com o desvio padrão dos valores.

Tabela 21 - Média dos Tempos de Atualização

	PySpark	PyFlink
Três Colunas	9,041	0,55
Vinte Colunas	1,601	0,783
TheBloke/Llama-2-7B-Chat-GGUF	3,164	0,896
TheBloke/Llama-2-13B-chat-GGUF	4,753	1,194
TheBloke/zephyr-7B-beta-GGUF	3,384	0,954

Fonte: criada pelo autor

Tabela 22 - Desvio Padrão dos Tempos de Atualização

	PySpark	PyFlink
Três Colunas	7,875	1,492
Vinte Colunas	3,072	6,382
TheBloke/Llama-2-7B-Chat-GGUF	7,459	4,754
TheBloke/Llama-2-13B-chat-GGUF	9,728	5,25
TheBloke/zephyr-7B-beta-GGUF	7,566	3,797

Fonte: criada pelo autor

Com isso, foi possível dizer que o PyFlink teve um desempenho melhor que o PySpark, com valores menores tanto de tempo médio de atualização gráfica quanto de desvio padrão. Isso corrobora com as percepções de que as atualizações foram

mais fluidas, além de indicar que o desempenho do PyFlink para esse cenário se apresenta como mais adequado.

Entretanto, existe a possibilidade de que o PyFlink apenas tenha prolongado o período de execução, de forma a encaixar essas atualizações de modo mais bem distribuído, o que significaria que existe uma perda de desempenho acentuada, apesar da percepção de maior fluidez. Considerando esse cenário, a Tabela 23 indica o tempo de execução total de cada fluxo de dados.

Tabela 23 - Tempo de Execução Total dos Fluxos

	PySpark	PyFlink
Três Colunas	298,359	275,068
Vinte Colunas	324,930	391,531
TheBloke/Llama-2-7B-Chat-GGUF	1072,58	330,656
TheBloke/Llama-2-13B-chat-GGUF	2134,13	543,494
TheBloke/zephyr-7B-beta-GGUF	1417,92	446,597

Fonte: criada pelo autor

A partir desse resultado, o PySpark se demonstra com um desempenho superior apenas para o caso das vinte colunas, com uma diferença de 66,601 segundos, ou, aproximadamente 83% do tempo necessário para o PyFlink. Em todos os outros cenários, o período de execução do PyFlink foi inferior a 50% do tempo necessário para o PySpark, o que significa uma maior fluidez sem perda de desempenho na execução.

Uma última análise foi feita sobre o tempo necessário para o início das atualizações gráficas. Esse resultado comparativo pode ser visto na Tabela 24.

Tabela 24 - Tempo de Início de Atualizações Gráficas

	PySpark	PyFlink
Três Colunas	35,509	33,7001
Vinte Colunas	35,773	143,192
TheBloke/Llama-2-7B-Chat-GGUF	126,517	89,6836

TheBloke/Llama-2-13B-chat-GGUF	184,759	104,628
TheBloke/zephyr-7B-beta-GGUF	144,544	77,1239

Fonte: criada pelo autor

Para esse comparativo, o PyFlink obteve pior desempenho também apenas no cenário das vinte colunas, sendo uma diferença mais expressiva. Apesar disso, o tempo de execução total não foi tão penalizado e permaneceu com a diferença de 66,601 segundos já citada anteriormente.

4.2.2 CONSUMO DE CPU E MEMÓRIA RAM

Para a análise do consumo de CPU e memória RAM, foram utilizadas as seguintes métricas, conforme imagem abaixo:

- O uso de CPU em porcentagem;
- O uso de memória RAM em porcentagem.

Isso foi necessário para medir o consumo computacional das duas tecnologias e perceber se existe uma diferença significativa nesse uso.

Foi considerada inicialmente também a medição em GB de RAM utilizados, porém, como esse valor precisa ser calculado, foi utilizada diretamente a porcentagem diretamente da biblioteca psutil, de forma a garantir que tanto CPU quanto RAM sejam analisados de acordo com suas porcentagens e sem cálculos intermediários.

As porcentagens foram coletadas em cada iteração e então foi realizada a média e o desvio padrão das amostras coletadas.

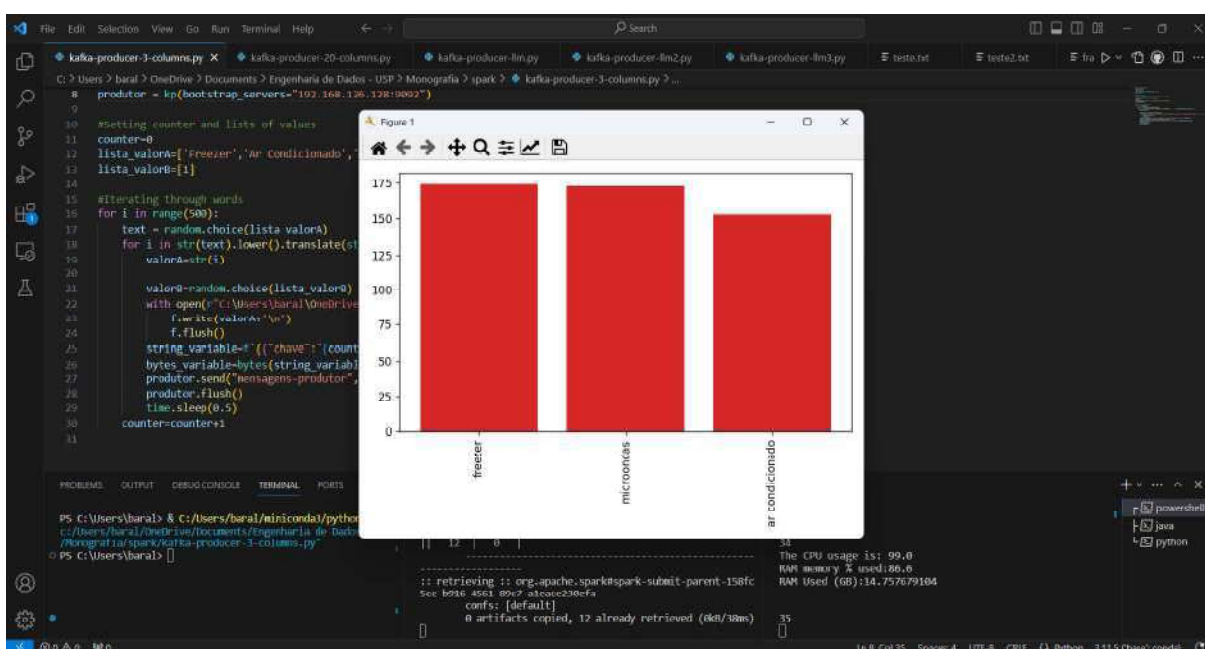
Apenas três exemplos (abordagens de três produtos, vinte e gerações com o modelo TheBloke/Llama-2-7B-Chat-GGUF) foram demonstrados a seguir, com o restante das visualizações e tabelas de resultados completos presentes no Apêndice B.

4.2.2.1 PYSPARK

Adiante, foram exibidos os gráficos e as métricas de média de porcentagem e desvio padrão dessa porcentagem tanto para uso de CPU quanto para uso memória RAM, de forma direcionada ao PySpark e sempre considerando o cenário com o tratamento gráfico.

4.2.2.1.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura 28 - Visualização Final



Fonte: criada pelo autor

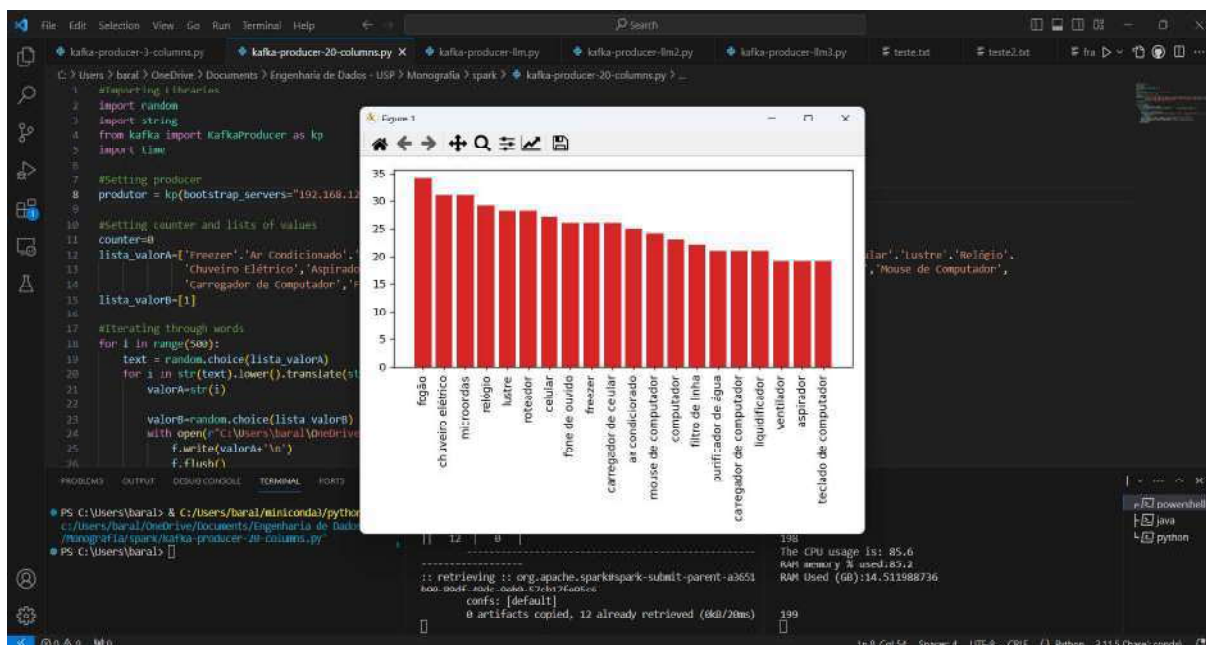
Tabela 25 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
58,333	50,000	86,675	0,558

Fonte: criada pelo autor

4.2.2.1.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura 29 - Visualização Final



Fonte: criada pelo autor

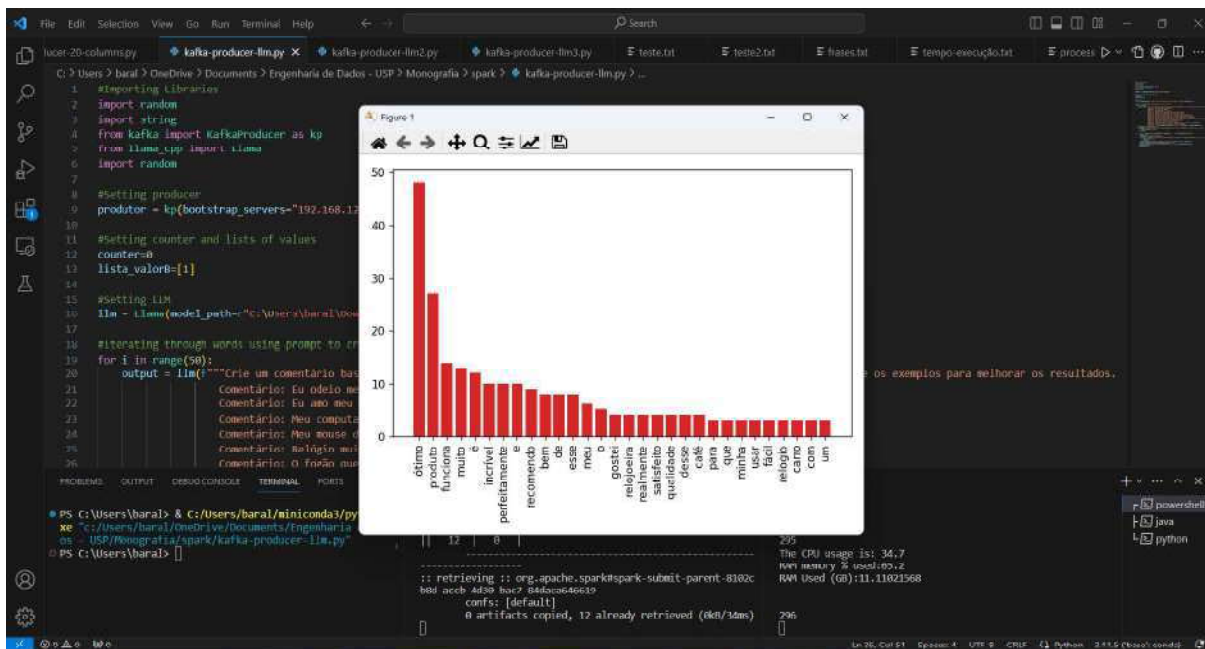
Tabela 26 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
56,688	49,556	85,372	0,986

Fonte: criada pelo autor

4.2.2.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura 30 - Visualização Final



Fonte: criada pelo autor

Tabela 27 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
62,626	48,461	88,813	6,986

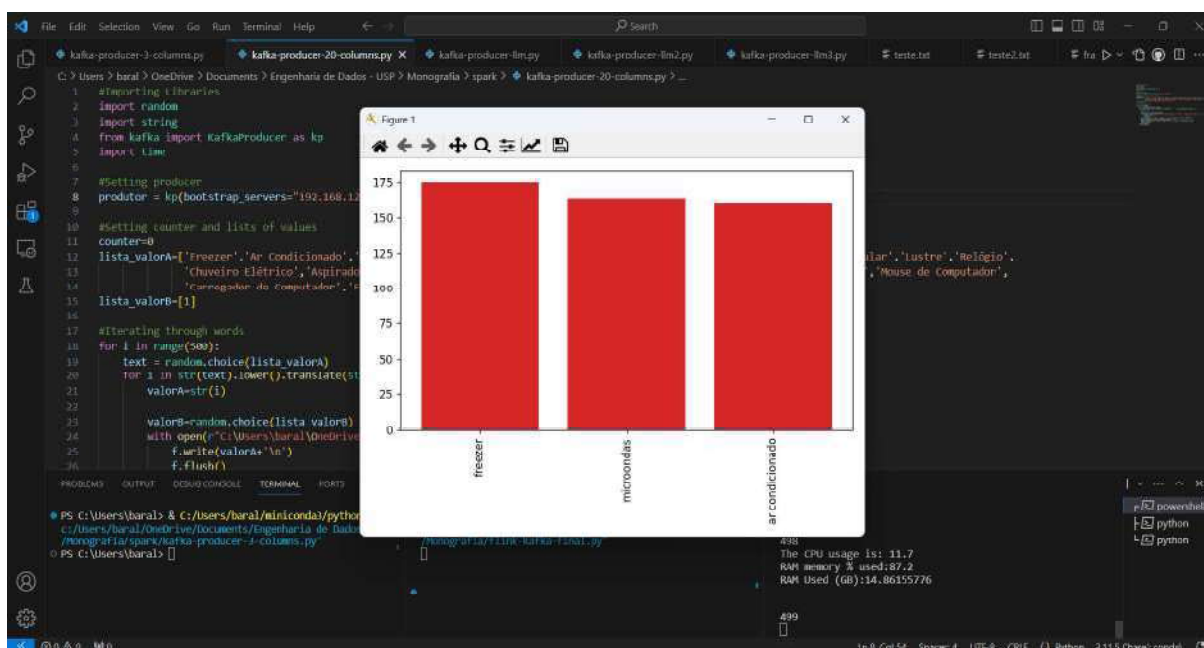
Fonte: criada pelo autor

4.2.2.2 PYFLINK

Da mesma forma, foram realizadas as coletas dos gráficos e das métricas para o PyFlink.

4.2.2.2.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura 31 - Visualização Final



Fonte: criada pelo autor

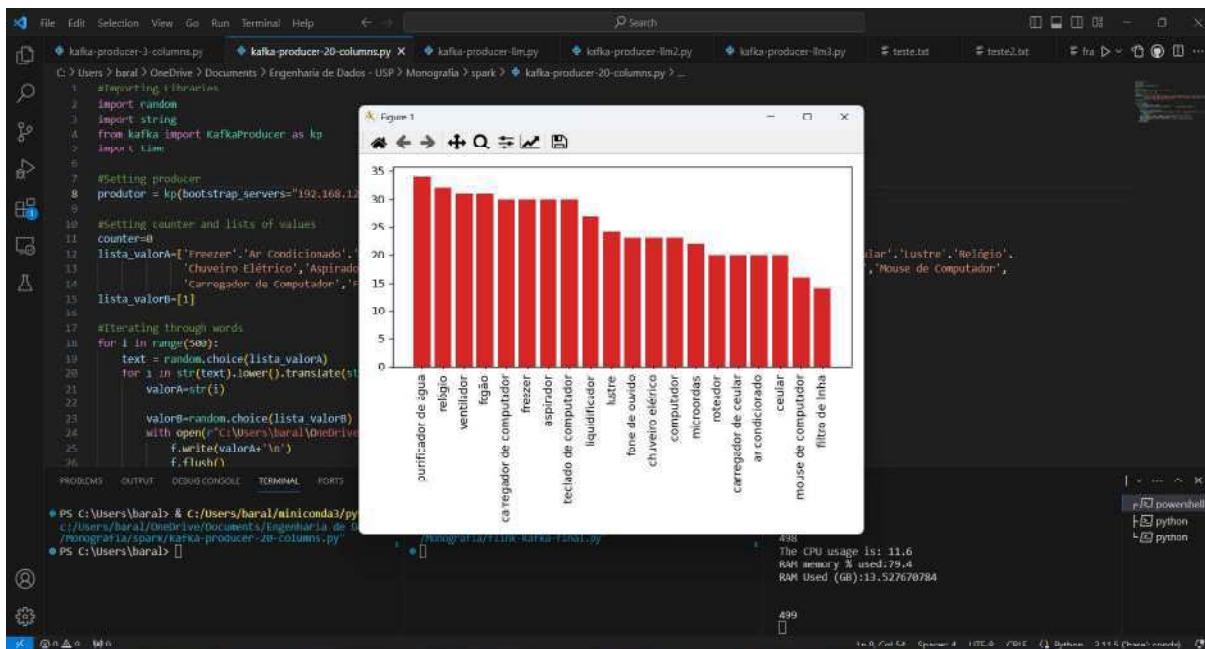
Tabela 28 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
3,976	16,591	85,179	0,570

Fonte: criada pelo autor

4.2.2.2.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura 32 - Visualização Final



Fonte: criada pelo autor

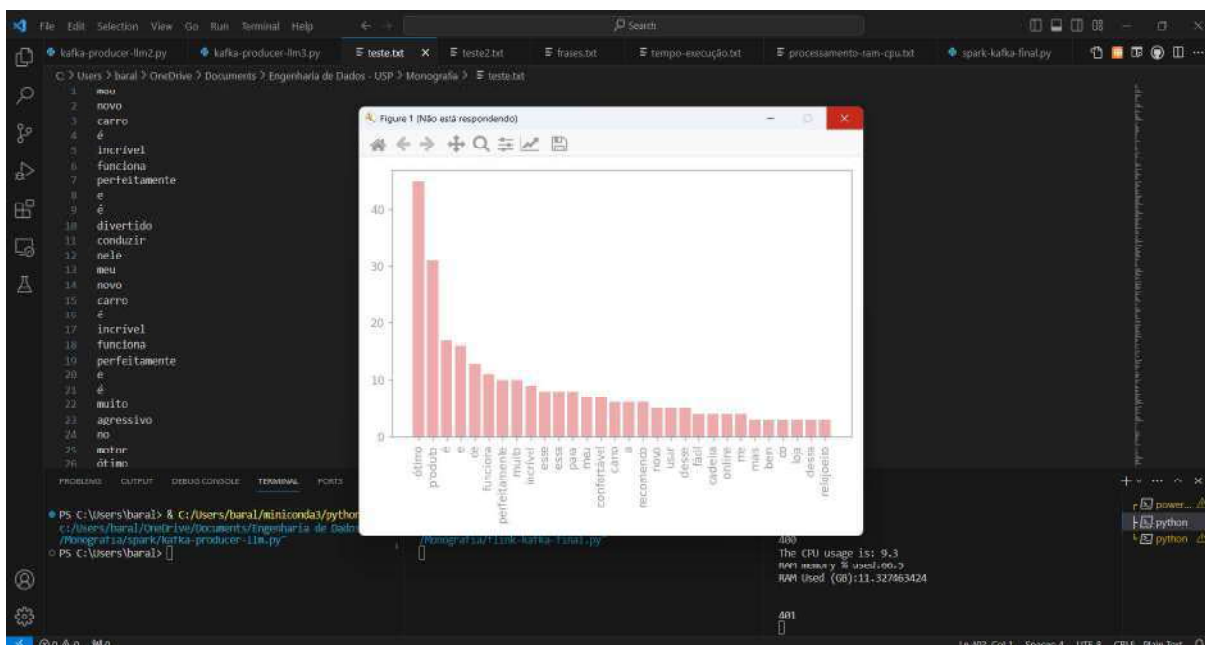
Tabela 29 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
4,279	17,778	80,057	0,675

Fonte: criada pelo autor

4.2.2.2.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura 33 - Visualização Final



Fonte: criada pelo autor

Tabela 30 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
16,073	32,768	97,730	3,454

Fonte: criada pelo autor

4.2.2.3 ANÁLISE DE USO DOS RECURSOS COMPUTACIONAIS

A partir das coletas parciais dos usos de CPU e RAM, conforme exemplos da Figura 30 e Tabela 27 (para o PySpark) e Figura 33 e Tabela 30 (para o PyFlink), com pontos percentuais, foi obtido o consolidado com a média na Tabela 31 e desvio padrão na Tabela 32, para o uso de CPU, e média na Tabela 33 e desvio padrão na Tabela 34, para o uso de RAM.

Tabela 31 - Média de Porcentagens de Uso de CPU

	PySpark	PyFlink
Três Colunas	58,333	3,976
Vinte Colunas	56,688	4,279
TheBloke/Llama-2-7B-Chat-GGUF	62,626	16,073
TheBloke/Llama-2-13B-chat-GGUF	71,197	20,223
TheBloke/zephyr-7B-beta-GGUF	70,184	10,059

Fonte: criada pelo autor

Tabela 32 - Desvio Padrão de Porcentagens de Uso de CPU

	PySpark	PyFlink
Três Colunas	50,000	16,591
Vinte Colunas	49,556	17,778
TheBloke/Llama-2-7B-Chat-GGUF	48,461	32,768
TheBloke/Llama-2-13B-chat-GGUF	45,032	35,643
TheBloke/zephyr-7B-beta-GGUF	45,741	23,752

Fonte: criada pelo autor

Tabela 33 - Média de Porcentagens de Uso de RAM

	PySpark	PyFlink
Três Colunas	86,675	85,179
Vinte Colunas	85,372	80,057
TheBloke/Llama-2-7B-Chat-GGUF	88,813	97,73
TheBloke/Llama-2-13B-chat-GGUF	96,734	97,452
TheBloke/zephyr-7B-beta-GGUF	91,344	96,036

Fonte: criada pelo autor

Tabela 34 - Desvio Padrão de Porcentagens de Uso de RAM

	PySpark	PyFlink
Três Colunas	0,558	0,57
Vinte Colunas	0,986	0,675
TheBloke/Llama-2-7B-Chat-GGUF	6,986	3,454
TheBloke/Llama-2-13B-chat-GGUF	5,886	1,316
TheBloke/zephyr-7B-beta-GGUF	6,254	4,59

Fonte: criada pelo autor

Com esses resultados da média e do desvio padrão do uso, em pontos percentuais, de CPU, ficou claro que o uso do PyFlink tem um consumo de CPU, em média, menor que o consumo do PySpark. Mesmo existindo períodos de máxima quantidade de CPU e mínima quantidade, a média do percentual de consumo de CPU do PyFlink ainda se manteve menor que o PySpark, indicando que pode haver um consumo de recurso computacional inferior.

Para o caso da Tabela 33 para média e Tabela 34 para desvio padrão, no uso da memória RAM, também em pontos percentuais, foi possível observar que tanto o PySpark quanto o PyFlink se apropriam de quase todo o recurso de RAM disponível para realizarem as execuções. Isso se verifica uma vez que durante as execuções, existem períodos em que a visualização gráfica de dados se torna irresponsiva. Também foi notado que o PyFlink possui um consumo ligeiramente maior de RAM para os casos dos Grandes Modelos de Linguagem e ligeiramente menor para os casos de *script* estático, com palavras pré-definidas mas escolhidas aleatoriamente. Essa diferença se torna insignificante para determinar uma variação expressiva entre as duas tecnologias, já que as duas se apropriam de valores acima de 80% de memória RAM e permanecem dessa forma durante as diferentes execuções, tendo variações entre si abaixo de 10%.

4.3 RESULTADO DOS GRANDES MODELOS DE LINGUAGEM

Para a análise dos Grandes Modelos de Linguagem, a Tabela 35 mostra os valores de tempo médio de atualização gráfica para os diferentes modelos, enquanto a Tabela 36 mostra o desvio padrão.

Tabela 35 - Tempo Médio de Atualização Gráfica

	TheBloke/Llama-2-7B-Chat-GGUF	TheBloke/Llama-2-13B-Chat-GGUF	TheBloke/zephyr-7B-beta-GGUF
PySpark	3,164	4,753	3,384
PyFlink	0,896	1,194	0,954

Fonte: criada pelo autor

Tabela 36 - Desvio Padrão de Tempo de Atualização Gráfica

	TheBloke/Llama-2-7B-Chat-GGUF	TheBloke/Llama-2-13B-Chat-GGUF	TheBloke/zephyr-7B-beta-GGUF
PySpark	7,459	9,728	7,566
PyFlink	4,754	5,25	3,797

Fonte: criada pelo autor

Além disso, a Tabela 37 mostra o tempo total de execução e a Tabela 38 mostra o tempo necessário para o início das atualizações gráficas.

Tabela 37 - Tempo Total de Execução

	TheBloke/Llama-2-7B-Chat-GGUF	TheBloke/Llama-2-13B-Chat-GGUF	TheBloke/zephyr-7B-beta-GGUF
PySpark	1072,58	2134,13	1417,92
PyFlink	330,656	543,494	446,597

Fonte: criada pelo autor

Tabela 38 - Tempo de Início de Atualizações Gráficas

	TheBloke/Llama-2-7B-Chat-GGUF	TheBloke/Llama-2-13B-Chat-GGUF	TheBloke/zephyr-7B-beta-GGUF
PySpark	126,517	184,759	144,544
PyFlink	89,6836	104,628	77,1239

Fonte: criada pelo autor

Esses resultados indicam que um modelo como o TheBloke/Llama-2-13B-chat-GGUF possui um tempo de processamento maior que os outros. Isso significa que em um cenário de *streaming* e no cenário com a máquina em questão, seu desempenho é mais inferior aos outros dois analisados.

Além disso, os modelos TheBloke/Llama-2-7B-Chat-GGUF e TheBloke/zephyr-7B-beta-GGUF possuem resultados bastante positivos e relativamente próximos em relação a tempo de atualização e tempo total de execução.

Análises de consumo de recursos computacionais e de qualidade dos textos gerados também foram realizadas para esse comparativo entre modelos.

Então, sobre os recursos, a Tabela 39 exibe o consumo médio de CPU por modelo enquanto a Tabela 40 exibe o desvio padrão. Os consumos médios em pontos percentuais da memória RAM pelos modelos também se apresentam na Tabela 41, que é acompanhada pela Tabela 42 com os respectivos desvios-padrões.

Tabela 39 - Média de Porcentagem de Uso de CPU

	TheBloke/Llama-2-7B-Chat-GGUF	TheBloke/Llama-2-13B-chat-GGUF	TheBloke/zephyr-7B-beta-GGUF
PySpark	62,626	71,197	70,184
PyFlink	16,073	20,223	10,059

Fonte: criada pelo autor

Tabela 40 - Desvio Padrão de Porcentagem de Uso de CPU

	TheBloke/Llama-2-7B-Chat-GGUF	TheBloke/Llama-2-13B-chat-GGUF	TheBloke/zephyr-7B-beta-GGUF
PySpark	48,461	45,032	45,741
PyFlink	32,768	35,643	23,752

Fonte: criada pelo autor

Tabela 41 - Média de Porcentagem de Uso de RAM

	TheBloke/Llama-2-7B-Chat-GGUF	TheBloke/Llama-2-13B-chat-GGUF	TheBloke/zephyr-7B-beta-GGUF
PySpark	88,813	96,734	91,344
PyFlink	97,73	97,452	96,036

Fonte: criada pelo autor

Tabela 42 - Desvio Padrão de Porcentagem de Uso de RAM

	TheBloke/Llama-2-7B-Chat-GGUF	TheBloke/Llama-2-13B-chat-GGUF	TheBloke/zephyr-7B-beta-GGUF
PySpark	6,986	5,886	6,254
PyFlink	3,454	1,316	4,59

Fonte: criada pelo autor

O consumo de CPU permaneceu bem baixo para o uso do PyFlink e também nos modelos de apenas 7 bilhões de parâmetros. Enquanto isso, o uso de memória RAM dos modelos foi muito próximo.

Em relação à qualidade das respostas e devido à imprevisibilidade dos modelos, foi realizada uma coletânea e análise das palavras dos três modelos para os exemplos executados na etapa de medição do tempo de execução, totalizando 100 gerações de texto para ambos PySpark e PyFlink. Palavras inexistentes ou gramaticalmente incorretas demonstram que o modelo não possui resultados tão satisfatórios. Além disso, foi feita uma análise nas frases geradas para verificar se possuem sentido ou algum erro.

Com isso, a Tabela 43 mostra o resultado das análises humanas feitas em cima das palavras e frases geradas, sendo que para a análise humana, foram revisadas todas as palavras geradas considerando-as como incorretas devido a termos como acentuação, inexistência, erros gramáticos, entre outros, porém sem considerar prolongamentos de palavras como incorretos (um exemplo é o da geração da palavra “amooo” no lugar de “amo”, não sendo considerada incorreta por se tratar de uma expressão popular). Para as frases também foram consideradas incoerentes as gerações de textos com palavras adicionais desnecessárias, erros gramaticais, textos sem sentido lógico, conjugação verbal incorreta, uso incorreto de gêneros, singular e plural, palavras ou termos inexistentes, acentuação, entre outros.

Tabela 43 - Proporção de Palavras e Frases Incorretas de Acordo com Avaliação Humana

	TheBloke/Llama-2-7B-Chat-GGUF	TheBloke/Llama-2-13B-chat-GGUF	TheBloke/zephyr-7B-beta-GGUF
Proporção de Palavras Incorretas	0,023	0,014	0,026
Proporção de Frases Incoerentes	0,39	0,24	0,51

Fonte: criada pelo autor

Considerando todos os resultados, para o cenário de *streaming*, os melhores modelos, que se adequam tanto em relação às questões de desempenho quanto à qualidade dos textos gerados são o TheBloke/Llama-2-7B-Chat-GGUF e o TheBloke/Llama-2-13B-chat-GGUF, esse último tendo desempenho de qualidade da geração de texto melhor, mas também possuindo um tempo de atualização gráfica maior devido à demora da inferência realizada na geração dos textos, o que pode dificultar seu uso contínuo em *streaming* (em que o tempo é algo crítico) na bancada e máquina utilizadas. Em situações de cadeias de *prompts*, por exemplo, o modelo de 13 bilhões de parâmetros poderia ter desempenho inferior ao de 7 bilhões.

5 CONCLUSÃO

Com esse trabalho, foi possível criar um fluxo de dados completo até a visualização utilizando diversas tecnologias atuais.

Também foi possível definir um tratamento gráfico de forma a contornar os desafios do cenário de *streaming* de dados que impactam a visualização de dados. Os tratamentos geraram uma visualização mais satisfatória e que pode ser melhor utilizada e monitorada, podendo ser usados como referência para cenários similares.

Esse estudo proporcionou um comparativo entre uma tecnologia que simula *streaming* através da abordagem de *micro-batches* (PySpark) e outra que trabalha efetivamente com *streaming* (PyFlink).

O PySpark mostrou-se melhor para um cenário específico de vinte colunas com palavras pré-definidas escolhidas aleatoriamente. Com exceção dessa situação, os outros casos demonstram que o PyFlink possui desempenho superior. Isso pode significar que é possível utilizar a tecnologia do PyFlink, que é *streaming* de fato no lugar do PySpark, que utiliza *micro-batches*, sem perdas de desempenho e inclusive podendo haver ganhos.

Por mais que o PySpark seja bastante popular atualmente, o PyFlink se provou robusto o suficiente para os cenários em questão e para substituí-lo na bancada estudada de forma ainda mais eficiente.

Além disso, o PyFlink se mostrou mais fluido em relação às atualizações gráficas quando comparado ao PySpark. O uso da visualização de dados como um todo foi favorecida com o uso da abordagem de *streaming* de fato em detrimento da abordagem de *micro-batches*.

Outro fator a ser mencionado é o consumo de recursos computacionais das duas tecnologias. Nesse estudo, o PyFlink demonstrou ser mais eficaz, conforme citado anteriormente, mantendo um menor consumo computacional.

Com isso, é possível dizer que a abordagem de *streaming* utilizada pelo PyFlink possui mais eficácia do que a de *micro-batches* do PySpark, já que em todos os comparativos, a abordagem de *streaming* foi superior.

Por último, o cenário de *streaming* pode ser explorado em relação aos diferentes modelos de inteligência artificial generativa que foram executados e posteriormente comparados de forma a se eleger modelos (TheBloke/Llama-2-7B-Chat-GGUF e TheBloke/Llama-2-13B-chat-GGUF), dentre os três utilizados, como mais adequados para condições específicas deste trabalho através do *streaming*. O de maior quantidade de parâmetros possuindo melhor geração textual e o de menor quantidade de parâmetros obtendo desempenho operacional (tempo de geração dos textos) melhor, embora a variação de tempo entre os dois seja próxima, caracterizando que o modelo de 13 bilhões poderia ser o melhor para o caso da bancada experimental.

Para todas as análises, criação da bancada e demais informações, foram utilizados os materiais descritos no Apêndice A. Além disso, todos os resultados com as respectivas figuras e tabelas se encontram no Apêndice B.

6 CONTRIBUIÇÕES DO TRABALHO

Esse trabalho proporcionou uma nova referência de tratamentos gráficos e exibição gráfica para o cenário de *streaming* de dados. Também demonstrou, de forma prática, como as tecnologias de *streaming* podem ser efetivamente robustas quando comparadas a tecnologias que simulam o *streaming*, como é o caso dos *micro-batches* e como uma poderia substituir a outra com possíveis ganhos, conforme observado nesse estudo. Por fim, esse trabalho forneceu um comparativo entre Grandes Modelos de Linguagem com métricas do cenário de *streaming* e avaliações dos textos gerados.

7 TRABALHOS FUTUROS

Como trabalho futuro, é possível explorar diferentes tipos de gráficos e ainda outras tecnologias de *streaming* para comparar os resultados e determinar se existem diferenças também, abordando conceitos de experiência do usuário nas visualizações geradas e elaborando pesquisas e entrevistas como formas de aplicação da metodologia de experiência do usuário para a verificação da melhoria das visualizações. Além disso, é possível realizar explorações em outros tipos de máquinas e diferentes bancadas para verificar o comportamento por parte de outros tipos de tecnologias, como Grandes Modelos de Linguagem Multimodais ou modelos mais recentes como o Mixtral-8x7B.

8 REFERÊNCIAS BIBLIOGRÁFICAS

NEVES, M. C. **What Are Quantized LLMs?** Disponível em: <<https://www.tensorops.ai/post/what-are-quantized-llms>>. Acesso em: 26 dez. 2023.

GERGANOV, G. llama.cpp. Disponível em: <<https://github.com/ggerganov/llama.cpp>>.

AKRAM, S. W., VARALAKSHMI, M., & SUDEEPTHI, J. (2019). **Streaming big data analytics-current status, challenges and connection of unbounded data processing platforms**. International Journal of Innovative Technology and Exploring Engineering, 8(9 Special Issue 2), 698–700. <https://doi.org/10.35940/ijitee.I1144.0789S219>

DASGUPTA, A., ARENDT, D. L., FRANKLIN, L. R., WONG, P. C., & COOK, K. A. (2018). **Human factors in streaming data analysis: Challenges and opportunities for information visualization**. Computer Graphics Forum, 37(1), 254–272. <https://doi.org/10.1111/cgf.13264>

RED HAT. **Arquitetura Orientada a Eventos**. Disponível em: <<https://www.redhat.com/pt-br/topics/integration/what-is-event-driven-architecture>>. Acesso em: 16 fev. 2024.

ILYUKHA, V. **Top 10 Big Data Frameworks in 2022**. Disponível em: <<https://jelvix.com/blog/top-5-big-data-frameworks>>. Acesso em: 28 dez. 2023.

Apache Kafka. Disponível em: <<https://kafka.apache.org/documentation/#gettingStarted>>.

CONFLUENT. **Streaming Data Pipelines**. Disponível em: <<https://www.confluent.io/learn/streaming-data-pipelines/>>. Acesso em: 29 dez. 2023.

CONCEPTS OF PROGRAMMING LANGUAGES TENTH EDITION. [s.l: s.n.]. Disponível em: <<https://www.ime.usp.br/~alvaroma/ucsp/proglang/book.pdf>>.

SEBESTA, R. W. Concepts of programming languages. Boston: Pearson, 2012.

Miniconda — miniconda documentation. Disponível em: <<https://docs.conda.io/projects/miniconda/en/latest/#>>. Acesso em: 25 dez. 2023.

IBM. **What Are Large Language models? | IBM.** Disponível em: <<https://www.ibm.com/topics/large-language-models>>. Acesso em: 28 dez. 2023.

IBM. **What Are Foundation models?** Disponível em: <<https://research.ibm.com/blog/what-are-foundation-models>>. Acesso em: 28 dez. 2023.

VMWARE. **Workstation Player : Run a Second, Isolated Operating System on a Single PC with VMware Workstation Player.** Disponível em: <<https://www.vmware.com/products/workstation-player.html>>.

UBUNTU. **Download Ubuntu Desktop | Download | Ubuntu.** Disponível em: <<https://ubuntu.com/download/desktop>>.

AMARAL, F. **Domine Apache Kafka, Fundamentos E Aplicações Reais.** Disponível em: <<https://www.udemy.com/course/aprenda-apache-kafka/#instructor-1>>. Acesso em: 25 dez. 2023.

TEAM, D. **Apache Kafka: O que é e como funciona?** Disponível em: <<https://blog.dp6.com.br/apache-kafka-o-que-%C3%A9-e-como-funciona-300a5736e388>>. Acesso em: 25 dez. 2023.

POWERS, D.; ARTHUR, D. **kafka-python — kafka-python 2.0.2-dev Documentation.** Disponível em: <<https://kafka-python.readthedocs.io/en/master/index.html>>. Acesso em: 25 dez. 2023.

Kafka Tutorial in Python. Disponível em: <<https://www.javatpoint.com/kafka-in-python>>. Acesso em: 25 dez. 2023.

Overview - Spark 3.0.0 Documentation. Disponível em: <<https://spark.apache.org/docs/latest/index.html>>.

Welcome to Spark Python API Docs! — PySpark 2.4.5 Documentation. Disponível em: <<https://spark.apache.org/docs/latest/api/python/index.html>>.

Structured Streaming + Kafka Integration Guide (Kafka Broker Version 0.10.0 or higher) - Spark 2.4.5 Documentation. Disponível em: <<https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html>>.

Structured Streaming Programming Guide - Spark 2.4.5 Documentation. Disponível em: <<https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>>

PANDEY, P. K.; LEARNING JOURNAL. **Apache Spark and Databricks - Stream Processing in Lakehouse.** Disponível em: <<https://www.udemy.com/course/spark-streaming-using-python/>>. Acesso em: 25 dez. 2023.

AMARAL, F. **Formação Spark Com Pyspark : O Curso Completo.** Disponível em: <<https://www.udemy.com/course/spark-curso-completo/>>. Acesso em: jul. 2023.

What Is Apache Flink? — Architecture #. Disponível em: <<https://flink.apache.org/what-is-flink/flink-architecture/>>. Acesso em: 25 dez. 2023.

Flink : Connectors : SQL : Kafka» 3.0.1-1.18. Disponível em: <<https://mvnrepository.com/artifact/org.apache.flink/flink-sql-connector-kafka/3.0.1-1.18>>. Acesso em: 25 dez. 2023.

PEDRO, J. **A Fast Look at Spark Structured Streaming + Kafka.** Disponível em: <<https://towardsdatascience.com/a-fast-look-at-spark-structured-streaming-kafka-f0ff64107325>>. Acesso em: 25 dez. 2023.

Welcome to Flink Python Docs! — PyFlink 1.18.dev0 Documentation. Disponível em: <<https://nightlies.apache.org/flink/flink-docs-release-1.18/api/python/>>. Acesso em: 25 dez. 2023.

BIG DATA LANDSCAPE. **Apache Flink & Kafka End-to-end Streaming Project Hands-on.** Disponível em:

<<https://www.udemy.com/course/flink-streaming-python-handson/>>. Acesso em: 25 dez. 2023.

KRAVCENKO, V. **Pipreqs - Generate requirements.txt File for Any Project Based on Imports**. Disponível em: <<https://github.com/bndr/pipreqs>>. Acesso em: 25 dez. 2023.

Miniconda — Miniconda Documentation. Disponível em: <<https://docs.conda.io/projects/miniconda/en/latest/>>. Acesso em: 25 dez. 2023.

Python Release Python 3.11.5. Disponível em: <<https://www.python.org/downloads/release/python-3115/>>. Acesso em: 25 dez. 2023.

Start Locally. Disponível em: <<https://pytorch.org/get-started/locally/#start-locally>>. Acesso em: 26 dez. 2023.

Matplotlib: Python Plotting — Matplotlib 3.1.1 Documentation. Disponível em: <<https://matplotlib.org/>>. Acesso em: 26 dez. 2023.

META RESEARCH. **Llama 2**. Disponível em: <<https://github.com/facebookresearch/llama>>. Acesso em: 26 dez. 2023.

PYAJUDEME. **Como Instalar Llama 2 No Windows (Python, ChatGPT, Meta, Facebook, Llama2, Bard)**. Disponível em: <<https://www.youtube.com/watch?v=5m7yntT5sfs>>. Acesso em: 26 dez. 2023.

PyTorch. Disponível em: <<https://pytorch.org/get-started/previous-versions/>>. Acesso em: 26 dez. 2023.

ANDREI. **Python Bindings for llama.cpp**. Disponível em: <<https://github.com/abetlen/llama-cpp-python>>. Acesso em: 26 dez. 2023.

GERGANOV, G. **llama.cpp**. Disponível em: <<https://github.com/ggerganov/llama.cpp>>. Acesso em: 26 dez. 2023.

JOBINS, T. **TheBloke/Llama-2-7B-Chat-GGUF · Hugging Face**. Disponível em: <<https://huggingface.co/TheBloke/Llama-2-7B-Chat-GGUF>>. Acesso em: 26 dez. 2023.

JOBINS, T. **TheBloke/Llama-2-13B-chat-GGUF · Hugging Face**. Disponível em: <<https://huggingface.co/TheBloke/Llama-2-13B-chat-GGUF>>. Acesso em: 26 dez. 2023.

JOBINS, T. **TheBloke/zephyr-7B-beta-GGUF · Hugging Face**. Disponível em: <<https://huggingface.co/TheBloke/zephyr-7B-beta-GGUF>>. Acesso em: 26 dez. 2023.

JOBINS, T. **TheBloke (Tom Jobbins)**. Disponível em: <<https://huggingface.co/TheBloke>>. Acesso em: 26 dez. 2023.

RODOLA, G. **psutil: Cross-platform Lib for Process and System Monitoring in Python**. Disponível em: <<https://pypi.org/project/psutil/>>. Acesso em: 26 dez. 2023.

RUPANISWEETY. **Psutil Module in Python**. Disponível em: <<https://www.geeksforgeeks.org/psutil-module-in-python/>>. Acesso em: 26 dez. 2023.

PYTHON SOFTWARE FOUNDATION. **Datetime — Basic Date and Time Types — Python 3.7.2 Documentation**. Disponível em: <<https://docs.python.org/3/library/datetime.html>>. Acesso em: 26 dez. 2023.

RATED_R_SUNDRAM. **How to Check the Execution Time of Python Script ?** Disponível em: <<https://www.geeksforgeeks.org/how-to-check-the-execution-time-of-python-script/>>. Acesso em: 26 dez. 2023.

RAF. **What Are Tokens and How to Count them?** Disponível em: <<https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-the-m>>. Acesso em: 26 dez. 2023.

Python String split() Method. Disponível em: <https://www.w3schools.com/python/ref_string_split.asp>. Acesso em: 26 dez. 2023.

How Could I Stop Printing Time Execution · Issue #999 · ggerganov/llama.cpp. Disponível em: <<https://github.com/ggerganov/llama.cpp/issues/999>>. Acesso em: 26 dez. 2023.

POWERS, D.; ARTHUR, D. **KafkaConsumer — kafka-python 2.0.2-dev Documentation.** Disponível em: <<https://kafka-python.readthedocs.io/en/master/apidoc/KafkaConsumer.html>>. Acesso em: 28 dez. 2023.

AssertionError When Using llama-cpp-python in Google Colab. Disponível em: <<https://stackoverflow.com/questions/76986412/assertionerror-when-using-llama-cpp-python-in-google-colab>>. Acesso em: 26 dez. 2023.

MANJEET_04. **Python | Remove Punctuation from String.** Disponível em: <<https://www.geeksforgeeks.org/python-remove-punctuation-from-string/>>. Acesso em: 26 dez. 2023.

REDAÇÃO. **Brasil registra 78,5 milhões de compras online.** Disponível em: <<https://tiinside.com.br/21/07/2021/brasil-registra-785-milhoes-de-compras-online/>>. Acesso em: 26 dez. 2023.

RITURAJSAHA. **File flush() Method in Python.** Disponível em: <<https://www.geeksforgeeks.org/file-flush-method-in-python/>>. Acesso em: 26 dez. 2023.

Performance — Matplotlib 3.8.2 Documentation. Disponível em: <<https://matplotlib.org/stable/users/explain/artists/performance.html>>. Acesso em: 26 dez. 2023.

SRINIVASA RAO POLADI. **Matplotlib 3.0 Cookbook over 150 Recipes to Create Highly Detailed Interactive Visualizations Using Python.** [s.l.] Birmingham Mumbai Packt October, 2018.

Working in Interactive Mode. Disponível em: <<https://subscription.packtpub.com/book/programming/9781789135718/1/ch01lvl1sec03/working-in-interactive-mode>>. Acesso em: 26 dez. 2023.

STACKOVERFLOW. Exception While Deleting Spark Temp Dir in Windows 7 64 Bit. Disponível em:

<<https://stackoverflow.com/questions/41825871/exception-while-deleting-spark-temp-dir-in-windows-7-64-bit>>. Acesso em: 26 dez. 2023.

SUMUKH. Exception While Deleting Spark Temp Dir in Windows 7 64 Bit. Disponível em:

<<https://stackoverflow.com/questions/41825871/exception-while-deleting-spark-temp-dir-in-windows-7-64-bit>>. Acesso em: 26 dez. 2023.

GOYAL, P. How to Get Current CPU and RAM Usage in Python? Disponível em: <<https://www.geeksforgeeks.org/how-to-get-current-cpu-and-ram-usage-in-python/>>. Acesso em: 26 dez. 2023.

JASAITIS, A. How to Extract Text after a Character in Spreadsheets | WPS Office Academy. Disponível em: <<https://www.wps.com/academy/how-to-extract-text-after-a-character-in-spreadsheet-s-quick-tutorials-1864747/>>. Acesso em: 26 dez. 2023.

FROLOV, A. 3 Ways to Remove Blank Rows in Excel - Quick Tip. Disponível em: <<https://www.ablebits.com/office-addins-blog/remove-blank-rows-in-excel/>>. Acesso em: 26 dez. 2023.

ZACH. Excel: Remove Duplicates but Keep Row with Max Value. Disponível em: <<https://www.statology.org/excel-remove-duplicates-keep-max/>>. Acesso em: 26 dez. 2023.

CHRIS, K. Sort Dictionary by Value in Python – How to Sort a Dict. Disponível em: <<https://www.freecodecamp.org/news/sort-dictionary-by-value-in-python/>>. Acesso em: 26 dez. 2023.

MANJEET_04. Python | N Largest Values in Dictionary. Disponível em: <<https://www.geeksforgeeks.org/python-n-largest-values-in-dictionary/>>. Acesso em: 26 dez. 2023.

Plotting Top 10 Values in Big Data. Disponível em:
<<https://stackoverflow.com/questions/72970343/plotting-top-10-values-in-big-data>>.
Acesso em: 26 dez. 2023.

SCOTTLITTLE. Rotate Axis Tick Labels. Disponível em:
<<https://stackoverflow.com/questions/10998621/rotate-axis-tick-labels>>. Acesso em:
27 dez. 2023.

BART. X-axis Label Gets Cut off of Graph. Disponível em:
<<https://stackoverflow.com/questions/33904163/x-axis-label-gets-cut-off-of-graph>>.
Acesso em: 27 dez. 2023.

APÊNDICE A - SCRIPTS EXECUTADOS E GITHUB COM MATERIAIS

Os códigos e materiais executados são descritos adiante, bem como o repositório com todos os materiais desenvolvidos durante o trabalho.

A.1 CÓDIGO PRODUTOR APACHE KAFKA COM TRÊS COLUNAS

```
1  #Importing Libraries
2  import random
3  import string
4  from kafka import KafkaProducer as kp
5  import time
6
7  #Setting producer
8  produtor = kp(bootstrap_servers="192.168.126.128:9092")
9
10 #Setting counter and lists of values
11 counter=0
12 lista_valorA=['Freezer','Ar Condicionado','Microondas']
13 lista_valorB=[1]
14
15 #Iterating through words
16 for i in range(500):
17     text = random.choice(lista_valorA)
18     for i in str(text).lower().translate(str.maketrans('', '', string.punctuation)).split('\n'):
19         valorA=str(i)
20
21         valorB=random.choice(lista_valorB)
22         with open(r"C:\Users\barai\OneDrive\Documents\Engenharia de Uados - USP\Monografia\teste.txt", 'a') as f:
23             f.write(valorA+'\n')
24             f.flush()
25         string_variable=f'{"chave": "{counter}", "valorA": "{valorA}", "valorB": "{valorB}"}'
26         bytes_variable=bytes(string_variable,encoding='utf-8')
27         produtor.send("mensagens-produtor",key=b'Chave %d' % counter, value=bytes_variable)
28         produtor.flush()
29         time.sleep(0.5)
30     counter=counter+1
```

A.2 CÓDIGO PRODUTOR APACHE KAFKA COM VINTE COLUNAS

```

1 #Importing Libraries
2 import random
3 import string
4 from kafka import KafkaProducer as kp
5 import time
6
7 #Setting producer
8 produtor = kp(bootstrap_servers="192.168.126.128:9092")
9
10 #Setting counter and lists of values
11 counter=0
12 lista_valorA=['Freezer','Ar Condicionado','Microondas','Computador','Ventilador','Fone de Ouvido','Celular','Carregador de Celular','Lustre','Relógio',
13              'Chuveiro Elétrico','Aspirador','Roteador','Fogão','Teclado de Computador','Liquidificador','Purificador de Água','Mouse de Computador',
14              'Carregador de Computador','Filtro de Linha']
15 lista_valorB=[1]
16
17 #Iterating through words
18 for i in range(500):
19     text = random.choice(lista_valorA)
20     for j in str(text).lower().translate(str.maketrans('', '', string.punctuation)).split('\n'):
21         valorA=str(i)
22
23         valorB=random.choice(lista_valorB)
24         with open(r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\teste.txt", 'a') as f:
25             f.write(valorA+'\n')
26             f.flush()
27             string_variable=f'{"chave":{counter},"valorA":{valorA},"valorB":{valorB}}'
28             bytes_variable=bytes(string_variable,encoding='utf-8')
29             produtor.send("mensagens-produtor",key=b'Chave id' % counter, value=bytes_variable)
30             produtor.flush()
31             time.sleep(0.5)
32         counter=counter+1

```

A.3 CÓDIGO PRODUTOR APACHE KAFKA COM THEBLOKE/LLAMA-2-7B-CHAT-GGUF

```

1 #Importing Libraries
2 import random
3 import string
4 from kafka import KafkaProducer as kp
5 from llama_cpp import Llama
6 import random
7
8 #Setting producer
9 produtor = kp(bootstrap_servers="192.168.126.128:9092")
10
11 #Setting counter and lists of values
12 counter=0
13 lista_valorB=[1]
14
15 #Setting LLM
16 llm = Llama(model_path="C:\Users\baral\Downloads\llama-2-7b-chat.Q4_K_M.gguf",verbose=False)
17
18 #Iterating through words using prompt to create comments
19 for i in range(50):
20     output = llm(prompt="Gere um comentário bastante curto positivo ou negativo sobre um produto que você comprou recentemente. Use os exemplos para melhorar os resultados:
21
22     Comentário: Eu odio meu liquidificador.
23     Comentário: Eu amo meu microondas.
24     Comentário: Meu computador é incrível.
25     Comentário: Meu mouse de computador não funciona muito bem.
26     Comentário: Relógio muito bom. Bem melhor que o meu antigo.
27     Comentário: O fogão que comprei já está com defeito. Não recomendo não.
28     Comentário: Melhor ar condicionado que já comprei. Funciona perfeitamente.
29     Comentário: Meu celular não funciona direito. Me arrependo de ter comprado.
30     Comentário: Comprei esse modelo de fone de ouvido ontem. Hoje eu abri a caixa com vários defeitos.
31     Comentário: Paguei muito caro desse freezer e não valeu a pena.
32     Comentário: Muito bom e barato esse purificador de água.
33     Comentário: Ventilador perfeito. Gostei bastante.
34     Comentário: Óchei! Tentei esse roteador. Muito barulhento e difícil de configurar.

```

```

25 Comentario: Relógio muito bom. Bem melhor que o meu antigo.
26 Comentario: O fogão que comprei já está com defeito. Não recomendo não.
27 Comentario: Melhor ar condicionado que já comprei. Funciona perfeitamente.
28 Comentario: Meu celular não funciona direito. Me arrependo de ter comprado.
29 Comentario: Comprei esse modelo de fone de ouvido ontem. Hoje eu abri e veio com vários defeitos.
30 Comentario: Paguei muito caro desse freezer e não valeu a pena.
31 Comentario: Muito bom e barato esse purificador de água.
32 Comentario: Ventilador perfeito. Gostei bastante.
33 Comentario: Achei terrível esse roteador. Muito barulhento e difícil de configurar.
34 Comentario: Achava que esse lustre seria melhor. Não gostei.
35 Comentario: "", max_tokens=30, stop=["Comentario:"], stream=False, repeat_penalty=1.15)
36 text=output["choices"][0]["text"]
37 with open(r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\frases.txt", 'a') as f:
38     f.write(str(text)+'\n\n')
39     f.flush()
40
41 for i in str(text).lower().translate(str.maketrans('', '', string.punctuation)).split():
42     valorA=str(i)
43     valorB=random.choice(lista_valorB)
44     with open(r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\taste.txt", 'a') as f:
45         f.write(valorA+'\n')
46         f.flush()
47     string_variable=f[{"chave": "{counter}", "valorA": "{valorA}", "valorB": "{valorB}"}]
48     bytes_variable=bytes(string_variable,encoding='utf-8')
49     produtor.send("mensagens-produtor",key='Chave %d' % counter, value=bytes_variable)
50     produtor.flush()
51     counter=counter+1

```

A.4 CÓDIGO PRODUTOR APACHE KAFKA COM THEBLOKE/LLAMA-2-13B-CHAT-GGUF

```

1 #Importing Libraries
2 import random
3 import string
4 from kafka import KafkaProducer as kp
5 from llama_cpp import Llama
6 import random
7
8 #Setting producer
9 produtor = kp(bootstrap_servers='192.168.126.128:9092')
10
11 #Setting counter and lists of values
12 counter=0
13 lista_valorB=[1]
14
15 #Setting LLM
16 llm = Llama(model_path="C:\Users\baral\Downloads\llama-2-13b-chat-Q4_K_M.gguf", verbose=False)
17
18 #Iterating through words using prompt to create comments
19 for i in range(50):
20     output = llm(f"""Crie um comentário bastante curto positivo ou negativo sobre um produto que você comprou recentemente. Use os exemplos para melhorar os resultados.
21     Comentario: Eu odeio meu liquidificador.
22     Comentario: Eu amo meu microondas.
23     Comentario: Meu computador é incrível.
24     Comentario: Meu mouse de computador não funciona muito bem.
25     Comentario: Relógio muito bom. Bem melhor que o meu antigo.
26     Comentario: O fogão que comprei já está com defeito. Não recomendo não.
27     Comentario: Melhor ar condicionado que já comprei. Funciona perfeitamente.
28     Comentario: Meu celular não funciona direito. Me arrependo de ter comprado.
29     Comentario: Comprei esse modelo de fone de ouvido ontem. Hoje eu abri e veio com vários defeitos.
30     Comentario: Paguei muito caro desse freezer e não valeu a pena.
31     Comentario: Muito bom e barato esse purificador de água.
32     Comentario: Ventilador perfeito. Gostei bastante.
33     Comentario: Achei terrível esse roteador. Muito barulhento e difícil de configurar.
34     Comentario: Achava que esse lustre seria melhor. Não gostei.

```

```

25         Comentário: Relógio muito bom. Bem melhor que o meu antigo.
26         Comentário: O fogão que comprei já está com defeito. Não recomendo não.
27         Comentário: Melhor ar condicionado que já comprei. Funciona perfeitamente.
28         Comentário: Meu celular não funciona direito. Me arrependo de ter comprado.
29         Comentário: Comprei esse modelo de fone de ouvido ontem. Hoje eu abri e veio com vários defeitos.
30         Comentário: Paguei muito caro desse freezer e não valeu a pena.
31         Comentário: Muito bom e barato esse purificador de água.
32         Comentário: Ventilador perfeito. Gostei bastante.
33         Comentário: Achei terrível esse roteador. Muito barulhento e difícil de configurar.
34         Comentário: Achava que esse lustre seria melhor. Não gostei.
35         Comentário: "", max_tokens=30, stop=["Comentário:"], stream=False, repeat_penalty=1.15)
36     text=output["choices"][0]["text"]
37     with open(r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\frases.txt", 'a') as f:
38         f.write(str(text)+'\n\n')
39         f.flush()
40
41     for i in str(text).lower().translate(str.maketrans('', '', string.punctuation)).split():
42         valorA=str(i)
43         valorB=random.choice(lista_valorB)
44         with open(r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\teste.txt", 'a') as f:
45             f.write(valorA+'\n')
46             f.flush()
47         string_variable=f'{{"chave":"counter"},"valorA":"{valorA}","valorB":"{valorB}"}}'
48         bytes_variable=bytes(string_variable,encoding='utf-8')
49         produtor.send("mensagens-produtor",key=b'Chave %d' % counter, value=bytes_variable)
50         produtor.flush()
51     counter=counter+1

```

A.5 CÓDIGO PRODUTOR APACHE KAFKA COM THEBLOKE/ZEPHYR-7B-BETA-GGUF

```

1 #Importing libraries
2 import random
3 import string
4 from kafka import KafkaProducer as kp
5 from llama_cpp import Llama
6 import random
7
8 #Setting producer
9 produtor = kp(bootstrap_servers="192.168.128:9092")
10
11 #Setting counter and lists of values
12 counter=0
13 lista_valorB=[1]
14
15 #Setting LLM
16 llm = Llama(model_path=r"C:\Users\baral\Downloads\zephyr-7b-beta.Q4_K_N.gguf",verbose=False)
17
18 #Iterating through words using prompt to create comments
19 for i in range(50):
20     output = llm(prompt="Crie um comentário bastante curto positivo ou negativo sobre um produto que você comprou recentemente. Use os exemplos para melhorar os resultados.")
21     Comentário: fu odio meu liquidificador.
22     Comentário: tu amo meu microondas.
23     Comentário: Meu computador é incrível.
24     Comentário: Meu mouse de computador não funciona muito bem.
25     Comentário: Relógio muito bom. Bem melhor que o meu antigo.
26     Comentário: O fogão que comprei já está com defeito. Não recomendo não.
27     Comentário: Melhor ar condicionado que já comprei. Funciona perfeitamente.
28     Comentário: Meu celular não funciona direito, me arrependo de ter comprado.
29     Comentário: Comprei esse modelo de fone de ouvido ontem. Hoje eu abri e veio com vários defeitos.
30     Comentário: Paguei muito caro desse freezer e não valeu a pena.
31     Comentário: Muito bom e barato esse purificador de água.
32     Comentário: Ventilador perfeito. Gostei bastante.
33     Comentário: Achei terrível esse roteador. Muito barulhento e difícil de configurar.
34     Comentário: Achava que esse lustre seria melhor. Não gostei.

```

```

25      Comentário: Relógio muito bom. Bem melhor que o meu antigo.
26      Comentário: O fogão que comprei já está com defeito. Não recomendo não.
27      Comentário: Melhor ar condicionado que já comprei. Funciona perfeitamente.
28      Comentário: Meu celular não funciona direito. Me arrependo de ter comprado.
29      Comentário: Comprei esse modelo de fone de ouvido ontem. Hoje eu abri e veio com vários defeitos.
30      Comentário: Paguei muito caro desse freezer e não valeu a pena.
31      Comentário: Muito bom e barato esse purificador de água.
32      Comentário: Ventilador perfeito. Gostei bastante.
33      Comentário: Achei terrível esse roteador. Muito barulhento e difícil de configurar.
34      Comentário: Achava que esse lustre seria melhor. Não gostei.
35      Comentário: """ , max_tokens=30, stop=["Comentário:"], stream=False, repeat_penalty=1.15)
36      text=output["choices"][0]["text"]
37      with open(r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\frases.txt", 'a') as f:
38          f.write(str(text)+'\n\n')
39          f.flush()
40
41      for i in str(text).lower().translate(str.maketrans(' ', ' ', string.punctuation)).split():
42          valorA=str(i)
43          valorB=random.choice(lista_valorB)
44          with open(r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\teste.txt", 'a') as f:
45              f.write(valorA+'\n')
46              f.flush()
47          string_variable=f'{{"chave": "{counter}", "valorA": "{valorA}", "valorB": "{valorB}"}}'
48          bytes_variable=bytes(string_variable,encoding='utf-8')
49          produtor.send("mensagens-produtor",key=b'Chave %d' % counter, value=bytes_variable)
50          produtor.flush()
51      counter=counter+1

```

A.6 CÓDIGO APACHE SPARK (PYSPARK)

```

1  #Importing libraries
2  from pyspark.sql import SparkSession
3  from pyspark.sql.functions import from_json, col
4  from pyspark.sql.types import StructType, StructField, StringType
5
6  #Starting job
7  if __name__ == "__main__":
8
9      #Setting SparkSession
10     spark = SparkSession \
11         .builder \
12         .appName("Streaming Monografia") \
13         .config("spark.streaming.stopGracefullyOnShutdown", "true") \
14         .getOrCreate()
15
16     #Setting schema
17     schema = StructType([
18         StructField("chave", StringType()),
19         StructField("valorA", StringType()),
20         StructField("valorB", StringType())
21     ])
22
23     #Setting kafka source
24     kafka_df = spark.readStream \
25         .format("kafka") \
26         .option("kafka.bootstrap.servers", "192.168.126.128:9092") \
27         .option("subscribe", "mensagens-produtor") \
28         .option("startingOffsets", "earliest") \
29         .load()
30
31
32     #Setting transformation process and dataframes
33     value_select_df = kafka_df.select(from_json(col("value").cast("string"), schema).alias("value"))

```

```

34
35     #Setting transformation process and dataframes
36     value_select_df = kafka_df.select(from_json(col("value").cast("string"), schema).alias("value"))
37     value_select_grouped_df = value_select_df.groupBy("valorA", "valorB", "valorB") .groupBy("valorA", "valorB") .agg({"valorB": "sum"}) withColumnRenamed("sum(valorB)", "valorB")
38     kafka_target_df = value_select_grouped_df.selectExpr("chave as key",
39                                                         """to_json(named_struct(
40                                                             valorA, valorA,
41                                                             valorB, valorB )) as value
42                                                         """)
43
44     #Setting target and writing
45     value_writestream_query = kafka_target_df \
46         .writeStream \
47         .queryName("Query Write Target") \
48         .format("kafka") \
49         .option("kafka.bootstrap.servers", "192.168.126.128:9092") \
50         .option("topic", "mensagens-consumidor") \
51         .outputMode("update") \
52         .option("checkpointLocation", r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\spark\data") \
53         .start()
54
55     #Setting to wait completion
56     value_writestream_query.awaitTermination()
57
58     #Command to submit
59     #spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0 "C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\spark\spark-kafka-final.py"

```

A.7 CÓDIGO APACHE FLINK (PYFLINK)

```

1 #Importing libraries
2 from pyflink.table import TableEnvironment, EnvironmentSettings
3 from pyflink.table.expressions import col
4
5 #Creating environment as streaming environment and creating a TableEnvironment
6 environment_settings = EnvironmentSettings.in_streaming_mode()
7 table_environment = TableEnvironment.create(environment_settings)
8
9 #Setting the jars to be used - here it is being used Apache Kafka, so there is a kafka connector
10 table_environment.get_config().get_configuration().set_string("pipeline.jars","file:///C:/Users/baral/Downloads/flink-sql-connector-kafka-3.0.1-1.18.jar")
11
12 #Creating the table that will be used as source
13 table_source = """CREATE TABLE table_source(chave VARCHAR, valorA VARCHAR, valorB INTEGER) WITH ('connector' = 'kafka', 'topic' = 'mensagens-producao',
14                                             'properties.bootstrap.servers' = '192.168.126.128:9092',
15                                             'properties.group.id' = 'consumidores', 'scan.startup.mode' = 'earliest-offset',
16                                             'format' = 'json')"""
17
18 #Creating the table that will be used as target (sink)
19 table_target = """CREATE TABLE table_target(valorA VARCHAR, valorB INTEGER, PRIMARY KEY (valorA) NOT ENFORCED) WITH ('connector' = 'upsert-kafka', 'topic' = 'mensagens-consumidor',
20                                                         'properties.bootstrap.servers' = '192.168.126.128:9092',
21                                                         'key.format' = 'json', 'value.format' = 'json')"""
22
23 #Running the sql statements to create source and target tables
24 table_environment.execute_sql(table_source)
25 table_environment.execute_sql(table_target)
26
27 #Creating sql statement that selects data from source
28 statement = "SELECT valorA,valorB FROM table_source"
29
30 #Running sql statementExecute the query and retrieve the result table
31 result_statement = table_environment.sql_query(statement).group_by(col('valorA')).select(col('valorA'),col('valorB').sum)
32

```

```

12 #Creating the table that will be used as source
13 table_source = """CREATE TABLE table_source(chave VARCHAR, valorA VARCHAR, valorB INTEGER) WITH ('connector' = 'kafka', 'topic' = 'mensagens-producao',
14                                             'properties.bootstrap.servers' = '192.168.126.128:9092',
15                                             'properties.group.id' = 'consumidores', 'scan.startup.mode' = 'earliest-offset',
16                                             'format' = 'json')"""
17
18 #Creating the table that will be used as target (sink)
19 table_target = """CREATE TABLE table_target(valorA VARCHAR, valorB INTEGER, PRIMARY KEY (valorA) NOT ENFORCED) WITH ('connector' = 'upsert-kafka', 'topic' = 'mensagens-consumidor',
20                                                         'properties.bootstrap.servers' = '192.168.126.128:9092',
21                                                         'key.format' = 'json', 'value.format' = 'json')"""
22
23 #Running the sql statements to create source and target tables
24 table_environment.execute_sql(table_source)
25 table_environment.execute_sql(table_target)
26
27 #Creating sql statement that selects data from source
28 statement = "SELECT valorA,valorB FROM table_source"
29
30 #Running sql statementExecute the query and retrieve the result table
31 result_statement = table_environment.sql_query(statement).group_by(col('valorA')).select(col('valorA'),col('valorB').sum)
32
33 #Inserting data to the target table
34 result_statement.execute_insert('table_target').wait()
35
36

```

A.8 CÓDIGO CONSUMIDOR KAFKA PARA ANÁLISE DE VISUALIZAÇÃO SEM TRATAMENTO GRÁFICO

```

1 #Importing libraries
2 from kafka import KafkaConsumer as kc
3 import matplotlib.pyplot as plt
4 import matplotlib.style as mplstyle
5 from ast import literal_eval
6 from datetime import datetime
7
8 #Setting Apache Kafka consumer
9 consumidor = kc("mensagens-consumidor",bootstrap_servers="192.168.126.128:9092",group_id="consumidores",auto_offset_reset='earliest')
10
11 #Initializing variables for chart
12 values = {}
13 x = []
14 y = values.values()
15
16 #Setting matplotlib to better performance
17 mplstyle.use('fast')
18
19 #Setting interactive mode in matplotlib
20 plt.ion()
21
22
23 #Setting figures, axis and chart type in matplotlib
24 fig = plt.figure()
25 ax = fig.add_subplot()
26
27
28
29
30 #Drawing figure and sending events
31 fig.canvas.draw()
32 fig.canvas.flush_events()
33

```

```

28
29
30 #Drawing figure and sending events
31 fig.canvas.draw()
32 fig.canvas.flush_events()
33
34
35
36 #Initializing counter
37 counter=0
38
39 #Iterating over messages in consumer
40 for mensagem in consumidor:
41
42     #Comitting
43     #consumidor.commit()
44
45
46
47     #Avoiding zeros and non-decoding values that generate errors
48     try:
49         variable_name = literal_eval(mensagem.value.decode('utf-8'))['valorA']
50         variable_value = int(literal_eval(mensagem.value.decode('utf-8'))['valorB'])
51     except:
52         variable_value=0
53
54     #Creating values accordingly
55     values[variable_name]=variable_value
56
57
58     #Cleaning and setting axis and xticks
59     ax.clear()
60     ax.bar(values.keys(),height=values.values(),color='tab:red')

```



```

58     #Cleaning and setting axis and xticks
59     ax.clear()
60     ax.bar(values.keys(),height=values.values(),color='tab:red')
61
62
63
64
65     #Registering the iteration
66     print(counter)
67
68     #Adding 1 to the counter
69     counter=counter+1
70     total=sum(values.values())
71     print(total)
72
73     with open(r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\teste2.txt", 'a') as f:
74         f.write(str(mensagem.value.decode('utf-8'))+'\n')
75         f.flush()
76
77
78
79     #Showing the data and adjusting
80     fig.canvas.draw()
81     fig.canvas.flush_events()
82
83
84
85
86

```

A.9 CÓDIGO CONSUMIDOR KAFKA PARA ANÁLISE DE VISUALIZAÇÃO COM TRATAMENTO GRÁFICO

```

1  #Importing libraries
2  from kafka import KafkaConsumer as kc
3  import matplotlib.pyplot as plt
4  import matplotlib.style as mplstyle
5  from ast import literal_eval
6  from datetime import datetime
7  from collections import Counter
8
9  #Setting Apache Kafka consumer
10 consumidor = kc("mensagens-consumidor",bootstrap_servers="192.168.126.128:9092",group_id="consumidores",auto_offset_reset="earliest")
11
12 #Initializing variables for chart
13 values = {}
14 x = []
15 y = values.values()
16
17 #Setting matplotlib to better performance
18 mplstyle.use("fast")
19
20 #Setting interactive mode in matplotlib
21 plt.ion()
22
23
24 #Setting figures, axis and chart type in matplotlib
25 fig = plt.figure()
26 ax = fig.add_subplot()
27
28
29
30
31 #Drawing figures and sending events
32 fig.canvas.draw()
33 fig.canvas.flush_events()
34 plt.tight_layout()

```

```

31 #Drawing figure and sending events
32 fig.canvas.draw()
33 fig.canvas.flush_events()
34 plt.tight_layout()
35
36
37
38 #Initializing counter
39 counter=0
40
41 #Iterating over messages in consumer
42 for mensagem in consumidor:
43     with open(r"C:\Users\hara1\OneDrive\Documents\Engenharia de Dados - IICP\Monografia\texto2.txt", 'a') as f:
44         f.write(str(mensagem.value.decode('utf-8'))+'\n')
45         f.flush()
46     #Avoiding zeros and non-decoding values that generate errors
47     try:
48         variable_name = literal_eval(mensagem.value.decode('utf-8'))['valorA']
49         variable_value = int(literal_eval(mensagem.value.decode('utf-8'))['valorB'])
50     except:
51         variable_value=0
52
53     #Sorting values accordingly
54     values[variable_name]=variable_value
55     values_ordered=sorted(values.items(), key=lambda x:x[1],reverse=True)
56     values_converted=dict(values_ordered)
57
58     #Setting top n values
59     n = 30
60     counter_values = Counter(values_converted)
61     top30 = dict(counter_values.most_common(n))
62
63
64

```

```

57
58     #Setting top n values
59     n = 30
60     counter_values = Counter(values_converted)
61     top30 = dict(counter_values.most_common(n))
62
63
64
65     #Cleaning and setting axis and xticks
66     ax.clear()
67     ax.bar(top30.keys(),height=top30.values(),color='tab:red')
68     plt.xticks(rotation=90)
69
70     #Showing the data and adjusting
71     fig.canvas.draw()
72     fig.canvas.flush_events()
73     plt.tight_layout()
74
75
76     #Registering the iteration
77     print(counter)
78
79     #Adding 1 to the counter
80     counter=counter+1
81

```

A.10 CÓDIGO CONSUMIDOR KAFKA PARA TEMPO DE EXECUÇÃO COM TRATAMENTO GRÁFICO

```

1 #Importing libraries
2 from kafka import KafkaConsumer as kc
3 import matplotlib.pyplot as plt
4 import matplotlib.style as mplstyle
5 from ast import literal_eval
6 from datetime import datetime
7 from collections import Counter
8
9 #Setting Apache Kafka consumer
10 consumidor = kc("mensagens-consumidor",bootstrap_servers="192.168.126.128:9092",group_id="consumidores",auto_offset_reset='earliest')
11
12 #Initializing variables for chart
13 values = {}
14 x = []
15 y = values.values()
16
17 #Setting matplotlib to better performance
18 mplstyle.use('fast')
19
20 #Setting interactive mode in matplotlib
21 plt.ion()
22
23
24 #Setting figures, axis and chart type in matplotlib
25 fig = plt.figure()
26 ax = fig.add_subplot()
27
28
29
30
31 #Drawing figures and sending events
32 fig.canvas.draw()
33 fig.canvas.flush_events()
34 plt.tight_layout()

```

```

31 #Drawing figure and sending events
32 fig.canvas.draw()
33 fig.canvas.flush_events()
34 plt.tight_layout()
35
36
37
38 #Initializing counter
39 counter=0
40
41 #Setting start time
42 start_counter = datetime.now()
43
44 #Iterating over messages in consumer0
45 for mensagem in consumidor:
46     with open("C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\tempo-execução.txt", 'a') as f:
47         end_counter = datetime.now()
48         delta = (end_counter - start_counter).total_seconds()
49         print("Tempo de Execução:"+str(delta)+" segundos\n")
50         f.write("Tempo de Execução:"+str(delta)+" segundos\n")
51         f.flush()
52
53
54
55
56     with open("C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\teste2.txt", 'a') as f:
57         f.write(str(mensagem.value.decode('utf-8'))+'\n')
58         f.flush()
59     #Avoiding zeros and non-decoding values that generate errors
60     try:
61         variable_name = literal_eval(mensagem.value.decode('utf-8'))['valorA']
62         variable_value = int(literal_eval(mensagem.value.decode('utf-8'))['valorB'])
63     except:
64         variable_value=0

```

```

55
56 with open(r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\teste2.txt", 'a') as f:
57     f.write(str(mensagem.value.decode('utf-8'))+'n')
58     f.flush()
59 #Avoiding zeros and non-decoding values that generate errors
60 try:
61     variable_name = literal_eval(mensagem.value.decode('utf-8'))['valorA']
62     variable_value = int(literal_eval(mensagem.value.decode('utf-8'))['valorB'])
63 except:
64     variable_value=0
65
66 #Sorting values accordingly
67 values[variable_name]=variable_value
68 values_ordered=sorted(values.items(), key=lambda x:x[1],reverse=True)
69 values_converted=dict(values_ordered)
70
71 #Setting top n values
72 n = 30
73 counter_values = Counter(values_converted)
74 top30 = dict(counter_values.most_common(n))
75
76
77
78 #Cleaning and setting axis and xticks
79 ax.clear()
80 ax.bar(top30.keys(),height=top30.values(),color='tab:red')
81 plt.xticks(rotation=90)
82
83 #Showing the data and adjusting
84 fig.canvas.draw()
85 fig.canvas.flush_events()
86 plt.tight_layout()
87

```

```

66 #Sorting values accordingly
67 values[variable_name]=variable_value
68 values_ordered=sorted(values.items(), key=lambda x:x[1],reverse=True)
69 values_converted=dict(values_ordered)
70
71 #Setting top n values
72 n = 30
73 counter_values = Counter(values_converted)
74 top30 = dict(counter_values.most_common(n))
75
76
77
78 #Cleaning and setting axis and xticks
79 ax.clear()
80 ax.bar(top30.keys(),height=top30.values(),color='tab:red')
81 plt.xticks(rotation=90)
82
83 #Showing the data and adjusting
84 fig.canvas.draw()
85 fig.canvas.flush_events()
86 plt.tight_layout()
87
88
89 #Registering the iteration
90 print(counter)
91
92 #Adding 1 to the counter
93 counter=counter+1
94

```

A.11 CÓDIGO CONSUMIDOR KAFKA PARA CONSUMO DE CPU E RAM COM TRATAMENTO GRÁFICO

```

1  #Importing libraries
2  from kafka import KafkaConsumer as kc
3  import matplotlib.pyplot as plt
4  import matplotlib.style as mplstyle
5  from ast import literal_eval
6  from datetime import datetime
7  from collections import Counter
8  import psutil
9  import time
10
11
12  #Setting Apache Kafka consumer
13  consumidor = kc("mensagens-consumidor",bootstrap_servers="192.168.126.126:9092",group_id="consumidores",auto_offset_reset='earliest')
14
15  #Initializing variables for chart
16  values = {}
17  x = []
18  y = values.values()
19
20  #Setting matplotlib to better performance
21  mplstyle.use('fast')
22
23  #Setting interactive mode in matplotlib
24  plt.ion()
25
26
27  #Setting figures, axis and chart type in matplotlib
28  fig = plt.figure()
29  ax = fig.add_subplot()
30
31

```

```

34  #Drawing figure and sending events
35  fig.canvas.draw()
36  fig.canvas.flush_events()
37  plt.tight_layout()
38
39
40
41  #initializing counter
42  counter=0
43
44
45
46  #iterating over messages in consumer
47  for message in consumidor:
48      with open("C:\\Users\\Useral\\Documents\\Engenharia de Dados - USP\\Monitoria\\processamento-ram-cpu.txt", "a") as f:
49          print('The CPU usage is: '+str(psutil.cpu_percent())+'\t')
50          print('RAM memory % used: '+str(psutil.virtual_memory()[2])+'\t')
51          print('RAM Used (GB): '+str(psutil.virtual_memory()[3]/1000000)+'\n\n')
52          f.write('The CPU usage is: '+str(psutil.cpu_percent())+'\t'+ 'RAM memory % used: '+str(psutil.virtual_memory()[2])+'\t'+ 'RAM Used (GB): '+str(psutil.virtual_memory()[3]/1000000)+'\n\n')
53          f.flush()
54
55
56
57
58
59
60
61  with open("C:\\Users\\Useral\\Documents\\Engenharia de Dados - USP\\Monitoria\\processamento-ram-cpu.txt", "a") as f:
62      f.write(str(message.value.decode('utf-8'))+'\n')
63      f.flush()
64  #holding zeros and non-decoding values that generate errors
65  try:
66      variable_name = literal_eval(message.value.decode('utf-8'))['valor']

```

```

60
61 with open(r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\teste2.txt", 'a') as f:
62     f.write(str(mensagem.value.decode('utf-8'))+'\n')
63     f.flush()
64 #Avoiding zeros and non-decoding values that generate errors
65 try:
66     variable_name = literal_eval(mensagem.value.decode('utf-8'))['valorA']
67     variable_value = int(literal_eval(mensagem.value.decode('utf-8'))['valorB'])
68 except:
69     variable_value=0
70
71 #Sorting values accordingly
72 values[variable_name]=variable_value
73 values_ordered=sorted(values.items(), key=lambda x:x[1],reverse=True)
74 values_converted=dict(values_ordered)
75
76 #Setting top n values
77 n = 30
78 counter_values = Counter(values_converted)
79 top30 = dict(counter_values.most_common(n))
80
81
82
83 #Cleaning and setting axis and xticks
84 ax.clear()
85 ax.bar(top30.keys(),height=top30.values(),color='tab:red')
86 plt.xticks(rotation=90)
87
88 #Showing the data and adjusting
89 fig.canvas.draw()
90 fig.canvas.flush_events()
91 plt.tight_layout()
92
93

```

```

75
76 #Setting top n values
77 n = 30
78 counter_values = Counter(values_converted)
79 top30 = dict(counter_values.most_common(n))
80
81
82
83 #Cleaning and setting axis and xticks
84 ax.clear()
85 ax.bar(top30.keys(),height=top30.values(),color='tab:red')
86 plt.xticks(rotation=90)
87
88 #Showing the data and adjusting
89 fig.canvas.draw()
90 fig.canvas.flush_events()
91 plt.tight_layout()
92
93
94 #Registering the iteration
95 print(counter)
96
97 #Adding 1 to the counter
98 counter=counter+1
99

```

A.12 GITHUB COM MATERIAIS, SCRIPTS E DEMAIS DOCUMENTOS DE ANÁLISES

Os arquivos desse trabalho podem ser acessados diretamente através do repositório do [GitHub](https://github.com/baraldi-daniel) presente no seguinte endereço:

<https://github.com/baraldi-daniel/Monograph>.

APÊNDICE B - RESULTADOS DAS VISUALIZAÇÕES E TABELAS

B.1 RESULTADOS DA VISUALIZAÇÃO

Resultados das visualizações gráficas sem tratamento e com tratamento, utilizando PySpark e PyFlink.

B.1.1 SEM TRATAMENTO

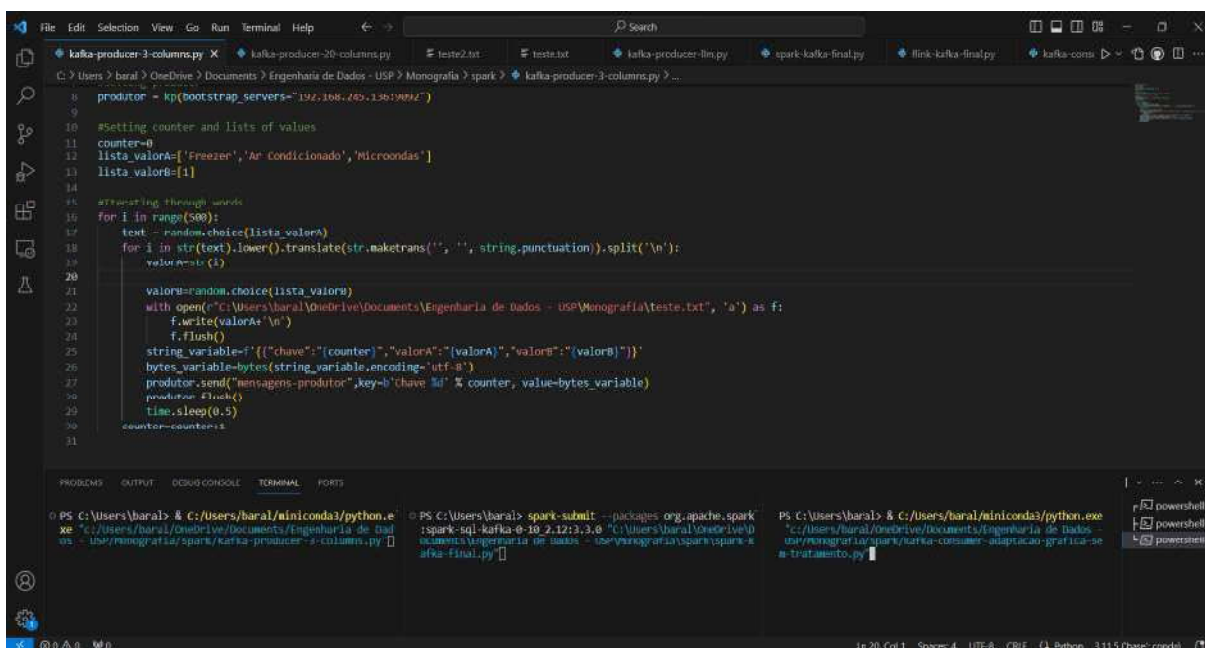
Visualizações sem tratamento para PySpark e PyFlink.

B.1.1.1 PYSPARK

Cinco abordagens utilizando PySpark.

B.1.1.1.1 TRÊS COLUNAS/PRODUTOS E SEM TRATAMENTO (500 EXEMPLOS)

Figura B-1 - Execução dos *Scripts*



```

8  produtor = kp(bootstrap_servers="192.168.162:13010002")
9
10 #setting counter and lists of values
11 counter=0
12 lista_valorA=['Freezer', 'Ar condicionado', 'Microondas']
13 lista_valorB=[1]
14
15 #travessia through words
16 for i in range(500):
17     text = random.choice(lista_valorA)
18     for i in str(text).lower().translate(str.maketrans('', '', string.punctuation)).split('\n'):
19         valorA=i
20
21         valorB=random.choice(lista_valorB)
22         with open(r"C:\Users\baral\OneDrive\Documents\Engenharia de Dados - USP\Monografia\teste.txt", 'a') as f:
23             f.write(valorA+' \n')
24             f.Flush()
25             string_variable=f'{"chave":{"counter"},"valorA":{"valorA"},"valorB":{"valorB"}}'
26             bytes_variable=bytes(string_variable,encoding='utf-8')
27             produtor.send("mensagens-produtor",key=b'chave id' % counter, value=bytes_variable)
28             producer.flush()
29             time.sleep(0.5)
30         counter=counter+1
31
  
```

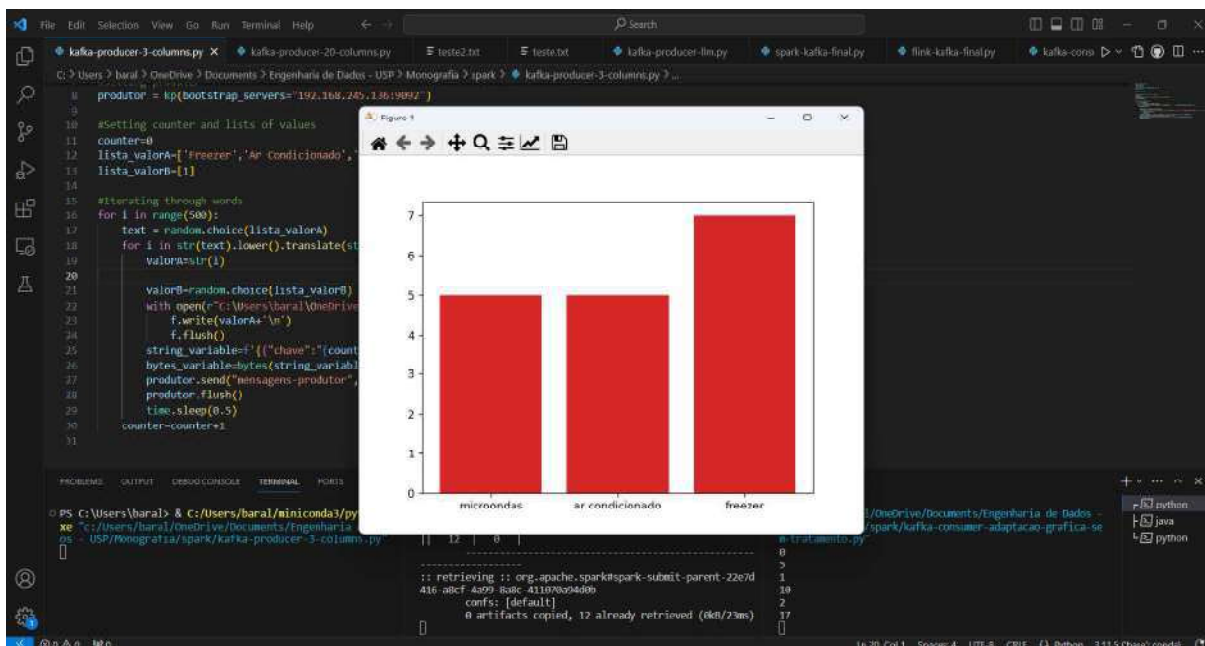
The terminal window shows the execution of the script using PySpark:

```

PS C:\Users\baral> C:/Users/baral/miniconda3/python.exe
xe "c:/Users/baral/OneDrive/Documents/Engenharia de Dad
os - usp/monografia/spark/kafka-producer-3-columns.py
PS C:\Users\baral> spark-submit --packages org.apache.spark
:spark-sql-kafka-0-10_2.12:3.0 "C:\Users\baral\OneDrive\D
ocuments\Engenharia de Dados - USP\monografia\spark\spark-s
afka-final.py
  
```

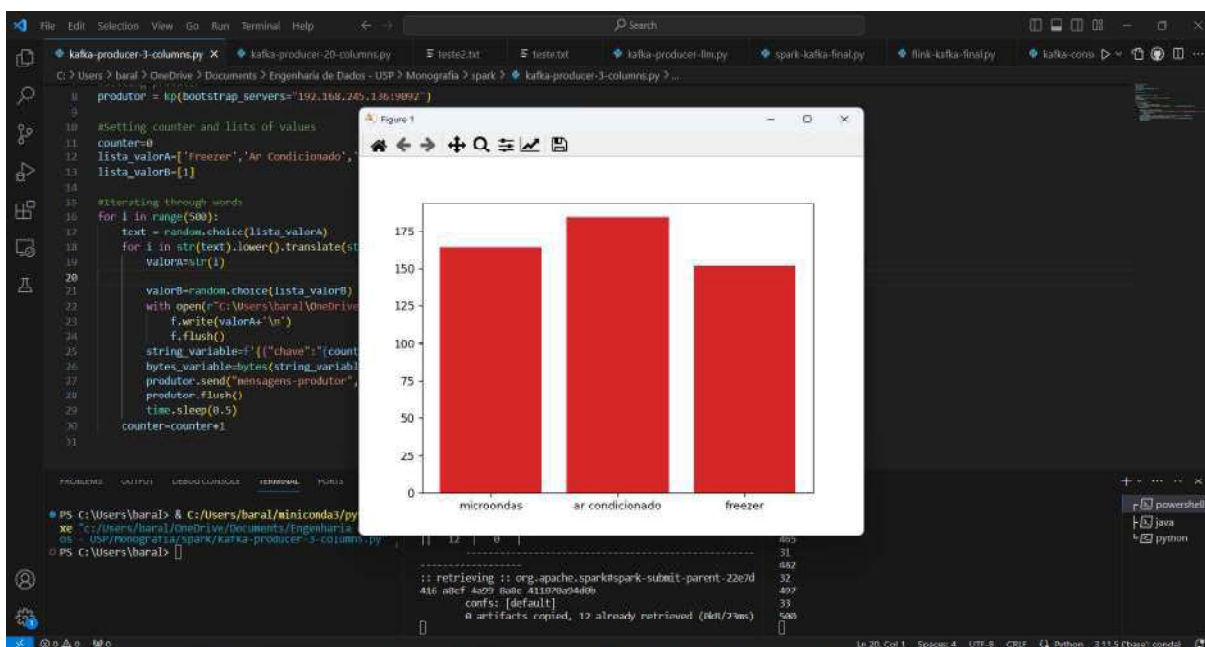
Fonte: criada pelo autor

Figura B-2 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-3 - Visualização Final



Fonte: criada pelo autor

Tabela B-1 - Amostras por Palavra

Gráfico de 3 Colunas	Contagem Total de Amostras	Contagem Freezer	Contagem Ar Condicionado	Contagem Microondas
	500	152	184	164

Fonte: elaborada pelo autor

B.1.1.1.2 VINTE COLUNAS/PRODUTOS E SEM TRATAMENTO (500 EXEMPLOS)

Figura B-4 - Execução dos Scripts

```

11 counter=0
12 lista_valores=['Freezer', 'Ar Condicionado', 'Microondas', 'Computador', 'Ventilador', 'Fone de Ouvido', 'Celular', 'Carregador de Celular', 'Lustre', 'Relógio',
13               'Chaveiro Elétrico', 'Aspirador', 'Roteador', 'Fogão', 'Teclado de computador', 'Liquidificador', 'Purificador de Água', 'Mouse de Computador',
14               'Carregador de computador', 'Filtro de Linha']
15 lista_valoresB=[1]
16
17 #Iterating through words
18 for i in range(500):
19     text = random.choice(lista_valores)
20     for j in str(text).lower().translate(str.maketrans('', '', string.punctuation)).split('\n'):
21         valorA=str(i)
22
23         valorB=random.choice(lista_valoresB)
24         with open("C:/Users/baral/OneDrive/Documents/Engenharia de Dados - USP/Monografia/teste.txt", 'a') as f:
25             f.write(valorA+'\n')
26             f.flush()
27             string_variable=f'{"chave": "{counter}", "valorA": "{valorA}", "valorB": "{valorB}"}'
28             bytes_variable=bytes(string_variable,encoding='utf-8')
29             producer.send("mensagens-producer",key='Chave %d' % counter, value=bytes_variable)
30             producer.flush()
31             time.sleep(0.5)
32             counter=counter+1
33

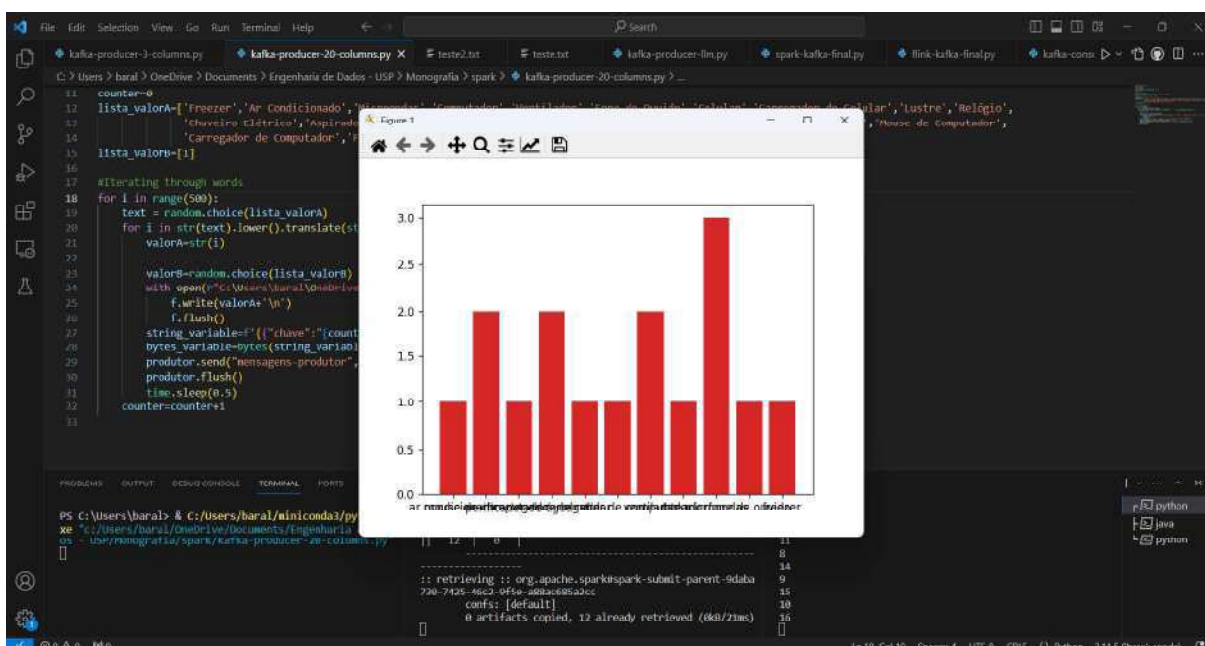
```

The terminal window shows the execution of the script in a PowerShell environment. The command executed is:

```
PS C:\Users\baral> & C:/Users/baral/miniconda3/python.exe "c:/Users/baral/OneDrive/Documents/Engenharia de Dados - USP/Monografia/spark/kafka-producer-20-columns.py"
```

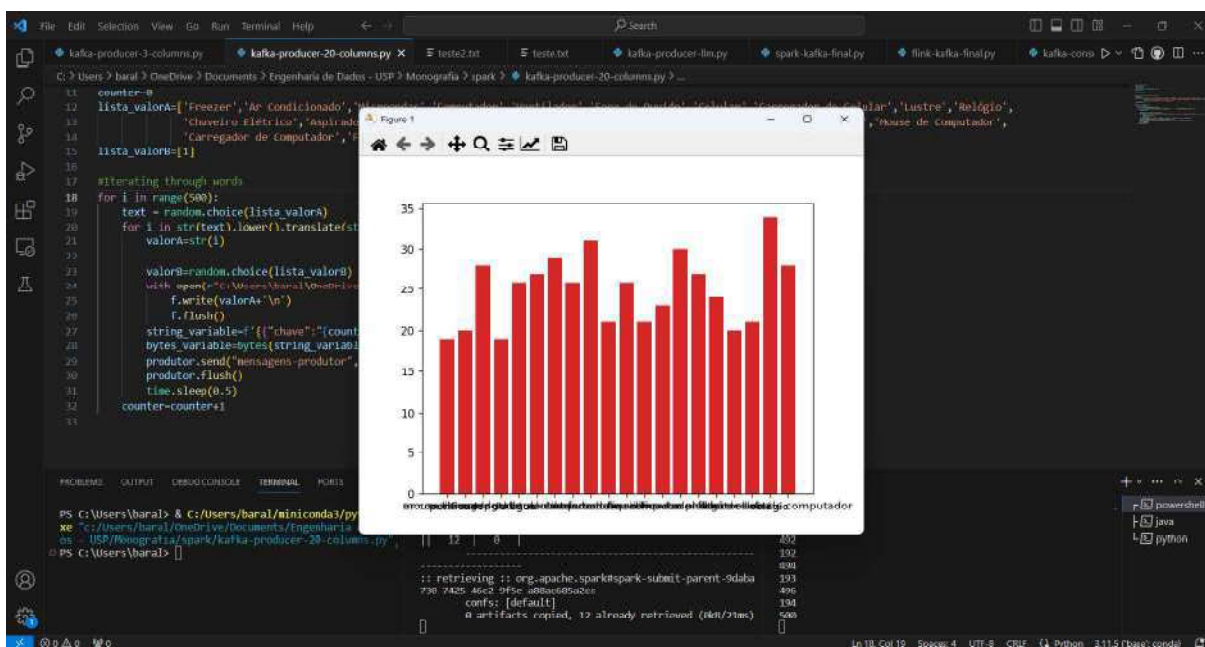
Fonte: criada pelo autor

Figura B-5 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-6 - Visualização Final



Fonte: criada pelo autor

Tabela B-2 - Contagem Total de Amostras com 500 Exemplos

Contagem Ar Condicionado	Contagem Aspirador	Contagem Carregador de Celular	Contagem Carregador de Computador	Contagem Celular
19	27	19	27	21

Contagem Chuveiro Elétrico	Contagem Computador	Contagem Filtro de Linha	Contagem Fogão	Contagem Fone de Ouvido
30	23	20	24	21

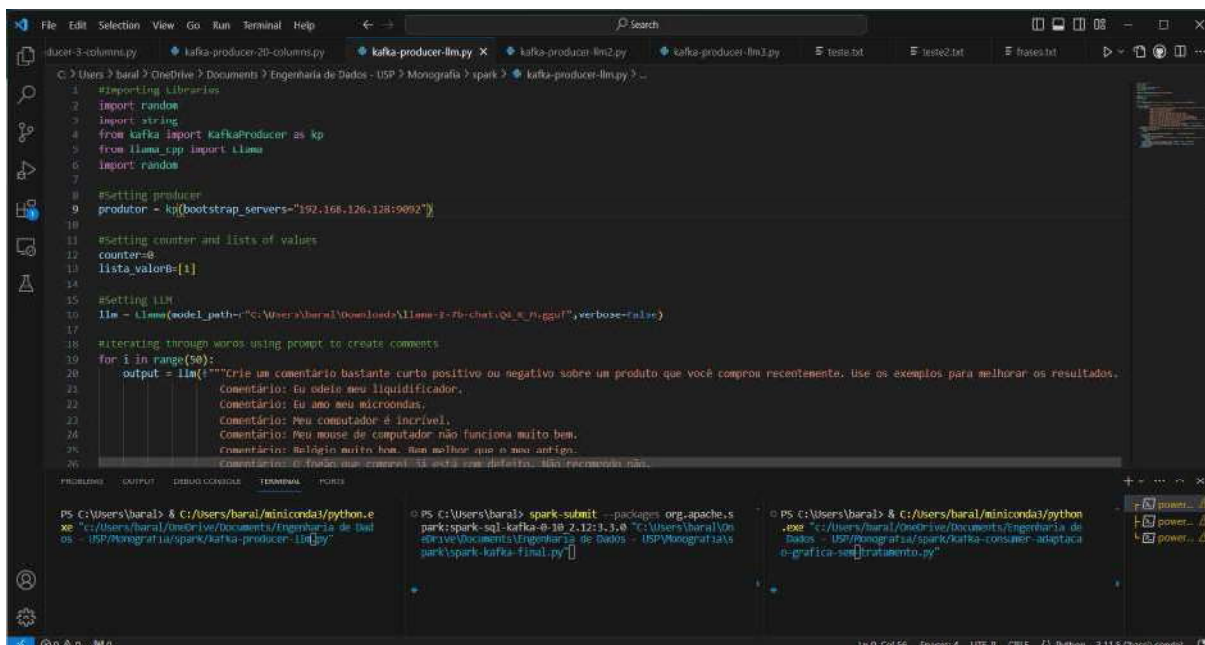
Contagem Freezer	Contagem Liquidificador	Contagem Lustre	Contagem Microondas	Contagem Mouse de Computador
26	21	26	31	20

Contagem Purificador de Água	Contagem Relógio	Contagem Roteador	Contagem Teclado de Computador	Contagem Ventilador
28	34	26	28	29

Fonte: criada pelo autor

B.1.1.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF E SEM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-7 - Execução dos Scripts



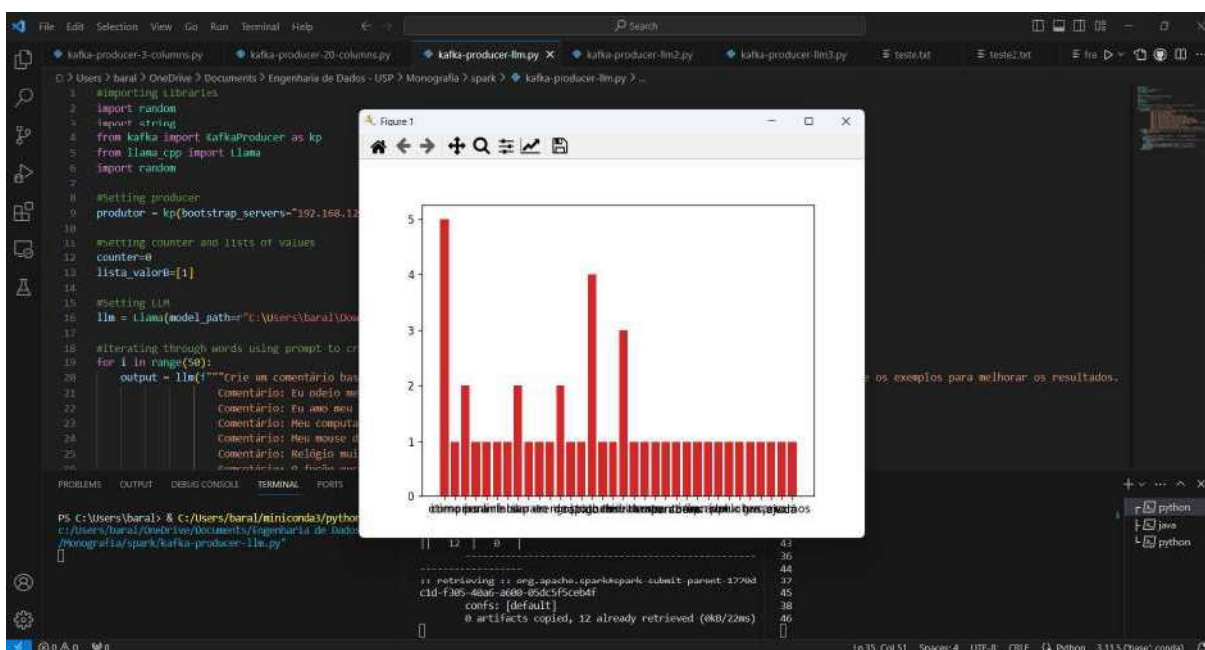
```

1 #reporting Libraries
2 import random
3 import string
4 from kafka import KafkaProducer as kp
5 from llama_cpp import Llama
6 import random
7
8 #setting producer
9 produtor = kp(bootstrap_servers="192.168.126.128:9092")
10
11 #setting counter and lists of values
12 counter=0
13 lista_valor=[1]
14
15 #setting LLM
16 llm = Llama(model_path="C:\Users\baral\Downloads\llama-2-7b-chat-gguf_2_7b_gguf", verbose=False)
17
18 #iterating through words using prompt to create comments
19 for i in range(50):
20     output = llm(prompt="Crie um comentário bastante curto positivo ou negativo sobre um produto que você compra recentemente. use os exemplos para melhorar os resultados.")
21     print(output)
22     #Comentário: Eu odeio meu liquidificador.
23     #Comentário: Eu amo meu microondas.
24     #Comentário: Meu computador é incrível.
25     #Comentário: Meu mouse de computador não funciona muito bem.
26     #Comentário: Relógio muito bom. Uma melhor que o meu antigo.
27     #Comentário: Este produto é muito bom e vale a pena comprar. Não recomendo mais.

```

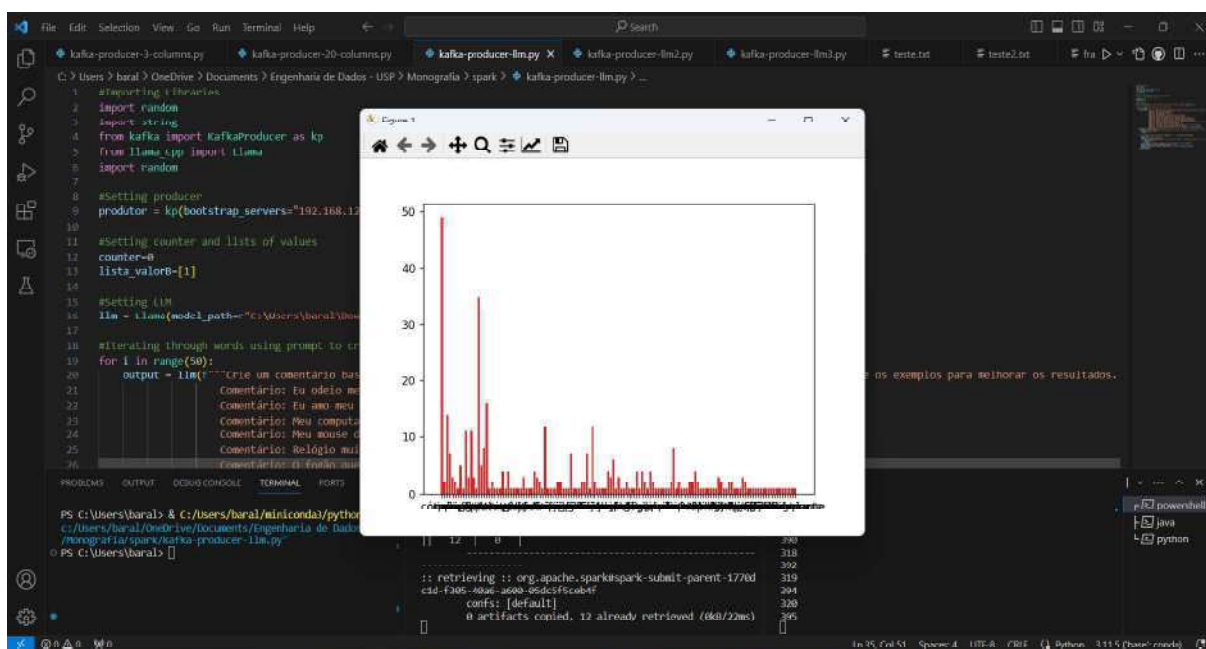
Fonte: criada pelo autor

Figura B-8 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-9 - Visualização Final



Fonte: criada pelo autor

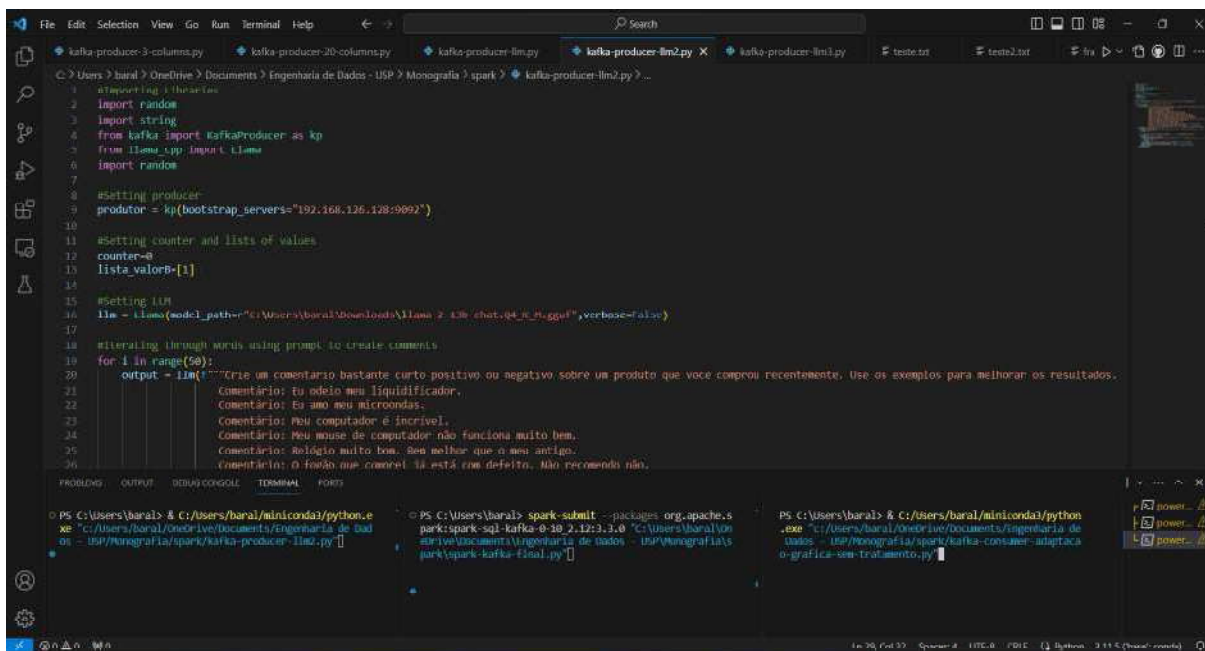
Tabela B-3 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	395	395

Fonte: criada pelo autor

B.1.1.1.4 THEBLOKE/LLAMA-2-13B-CHAT-GGUF E SEM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-10 - Execução dos Scripts



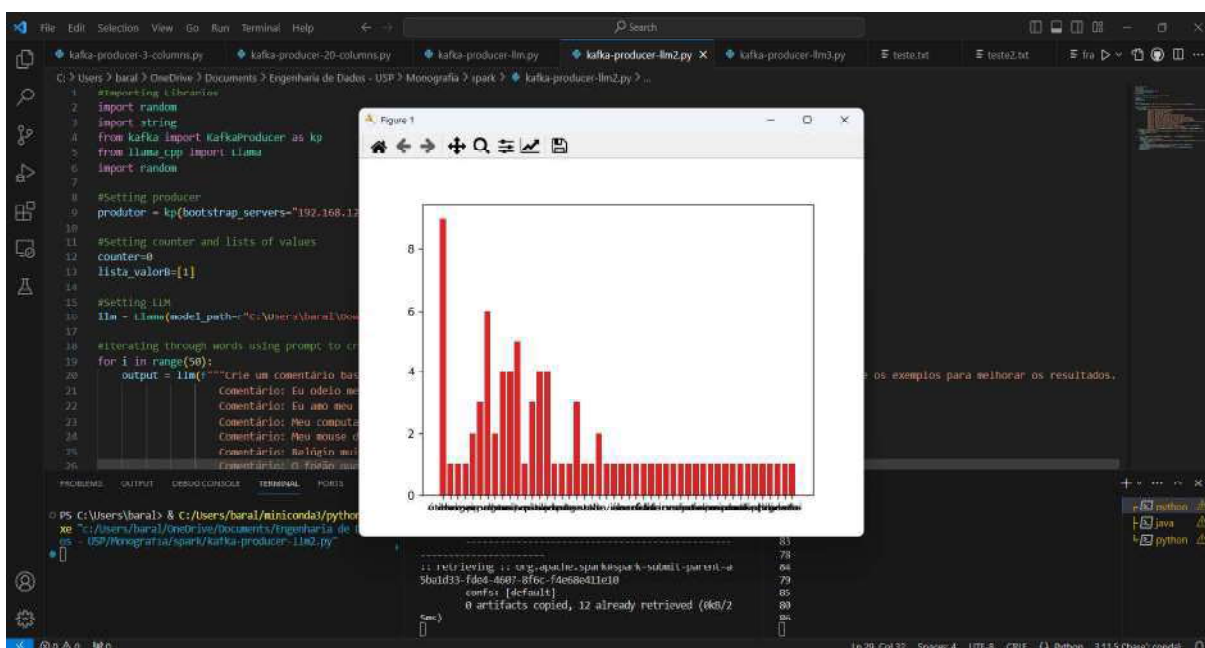
```

1 #Importing Libraries
2 import random
3 import string
4 from kafka import KafkaProducer as kp
5 from llama_cpp import Llama
6 import random
7
8 #Setting producer
9 produtor = kp(bootstrap_servers="192.168.128.9092")
10
11 #Setting counter and lists of values
12 counter=0
13 lista_valorB=[1]
14
15 #Setting LLM
16 llm = Llama(model_path="C:\Users\baral\Downloads\llama-2-13b-chat-gguf", verbose=False)
17
18 iterating through words using prompt to create comments
19 for i in range(50):
20     output = llm(prompt="Crie um comentário bastante curto positivo ou negativo sobre um produto que voce comprou recentemente. Use os exemplos para melhorar os resultados.
21     Comentário: tu odeio meu liquidificador,
22     Comentário: Eu amo meu microondas,
23     Comentário: Meu computador é incrível,
24     Comentário: Meu mouse de computador não funciona muito bem,
25     Comentário: Relógio muito bom, Sem melhor que o meu antigo,
26     Comentário: O fone de ouvido que comprei tá está com defeito, Não recomendo não.")

```

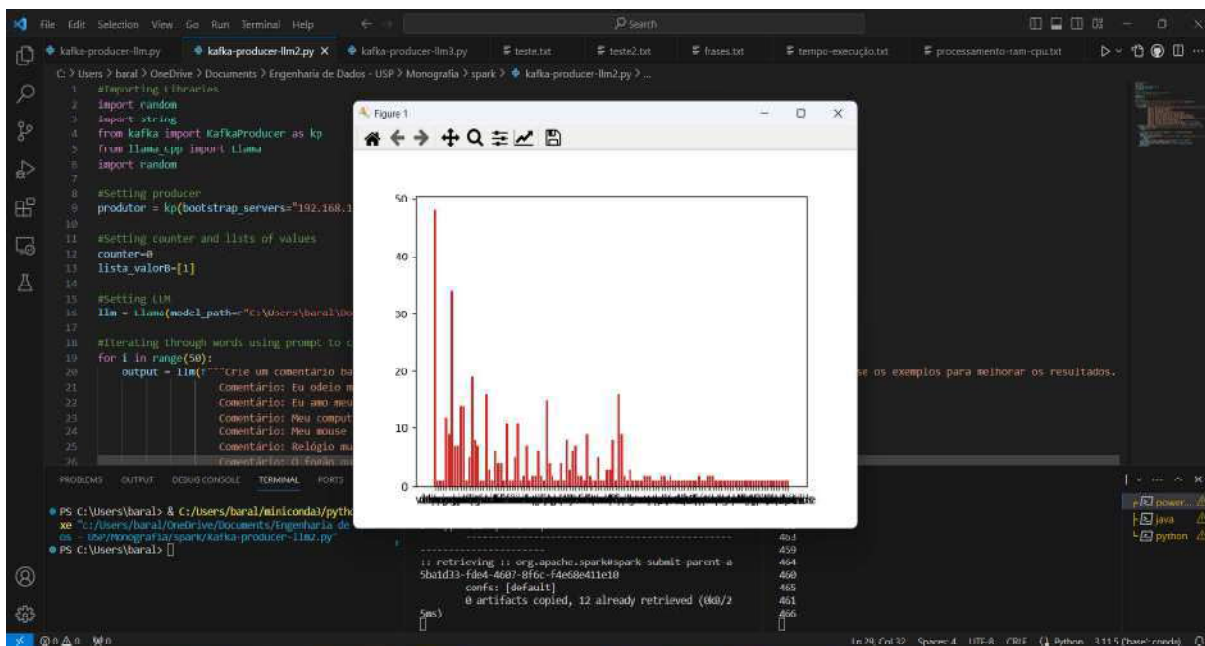
Fonte: criada pelo autor

Figura B-11 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-12 - Visualização Final



Fonte: criada pelo autor

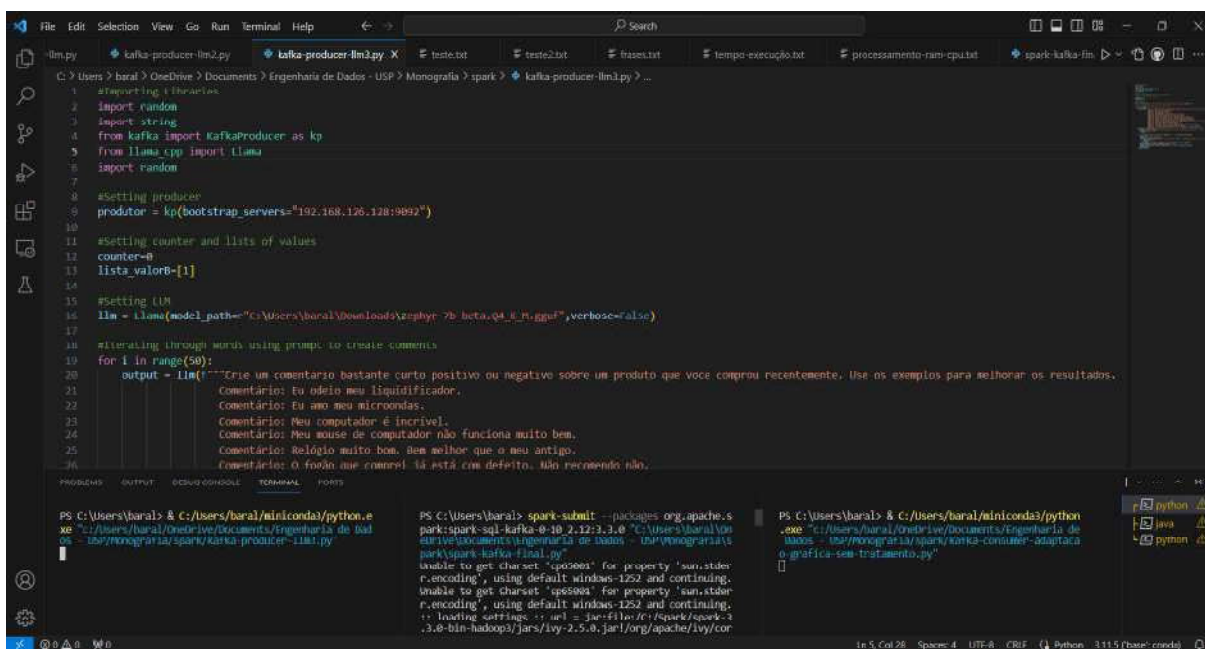
Tabela B-4 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	466	466

Fonte: criada pelo autor

B.1.1.1.5 THEBLOKE/ZEPHYR-7B-BETA-GGUF E SEM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-13 - Execução dos Scripts



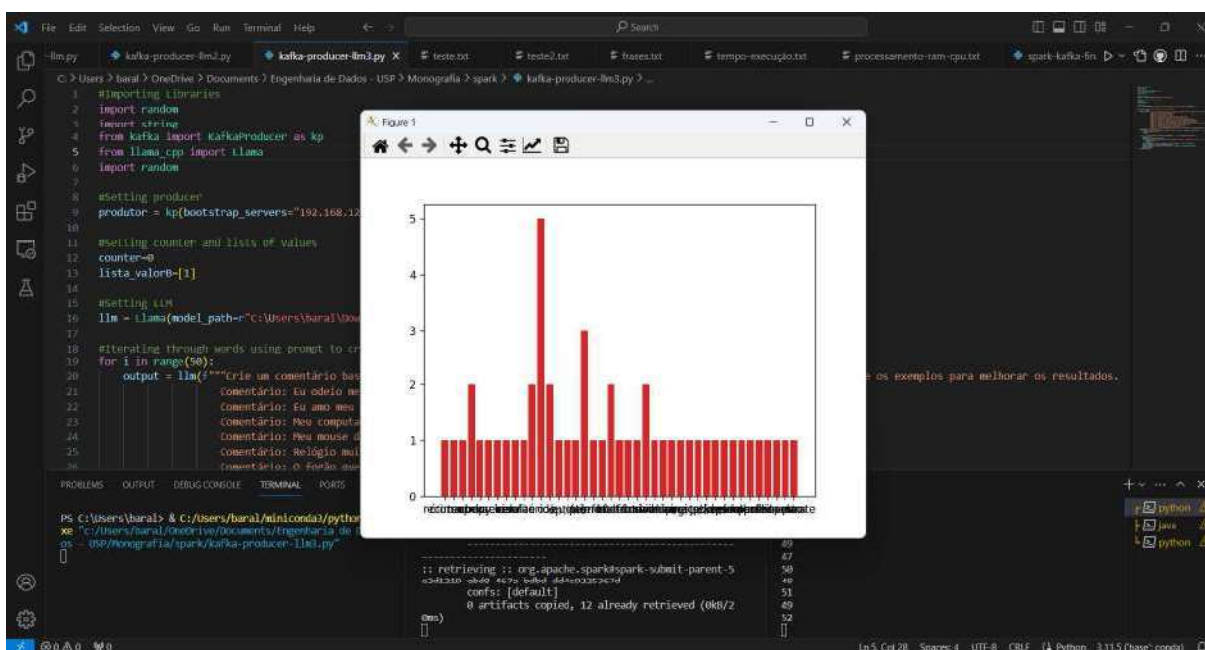
```

1 #importing libraries
2 import random
3 import string
4 from kafka import KafkaProducer as kp
5 from llama_cpp import Llama
6 import random
7
8 #setting producer
9 produtor = kp(bootstrap_servers="192.168.128:9092")
10
11 #setting counter and lists of values
12 counter=0
13 lista_valor0=[1]
14
15 #setting LLM
16 llm = Llama(model_path="C:\Users\baral\Downloads\zephyr-7b-beta-gguf",verbose=False)
17
18 #iterating through words using prompt to create comments
19 for i in range(50):
20     output = llm("Crie um comentário bastante curto positivo ou negativo sobre um produto que voce comprou recentemente. Use os exemplos para melhorar os resultados.
21     Comentário: Eu odeio meu liquidificador.
22     Comentário: Eu amo meu microondas.
23     Comentário: Meu computador é incrível.
24     Comentário: Meu mouse de computador não funciona muito bem.
25     Comentário: Relógio muito bom, mas melhor que o meu antigo.
26     Comentário: O fone que comrei tá está com defeito. Não recomendo não.")

```

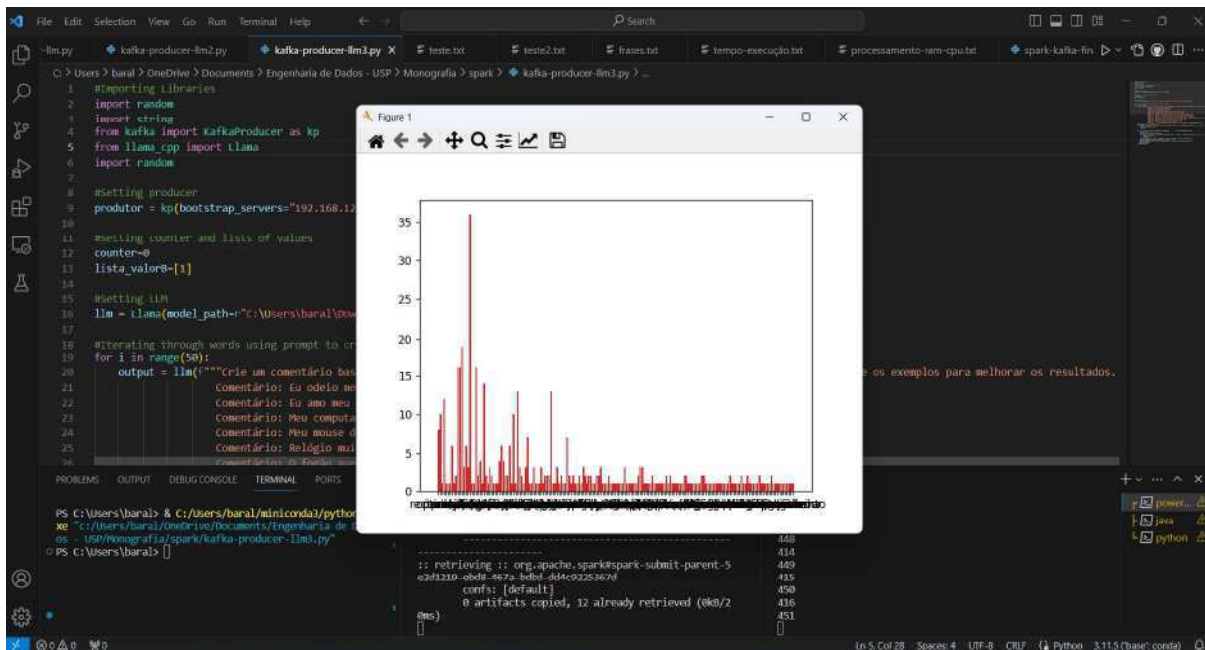
Fonte: criada pelo autor

Figura B-14 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-15 - Visualização Final



Fonte: criada pelo autor

Tabela B-5 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	451	451

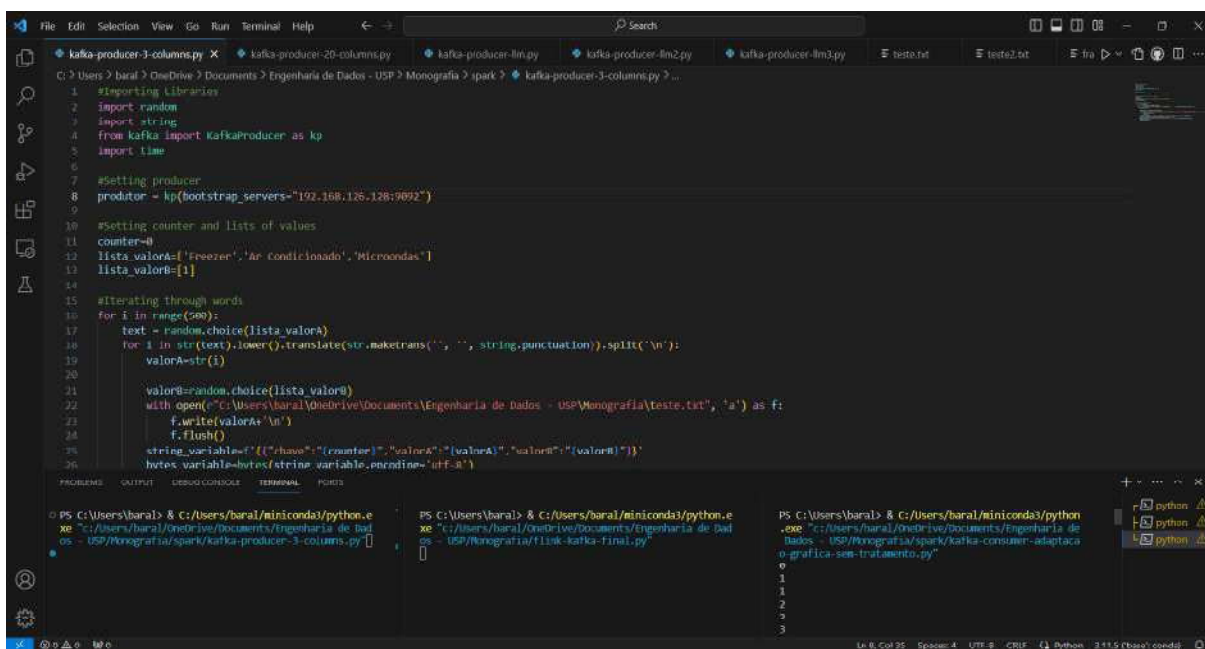
Fonte: criada pelo autor

B.1.1.2 PYFLINK

Cinco abordagens utilizando PyFlink.

B.1.1.2.1 TRÊS COLUNAS/PRODUTOS E SEM TRATAMENTO (500 EXEMPLOS)

Figura B-16 - Execução dos *Scripts*



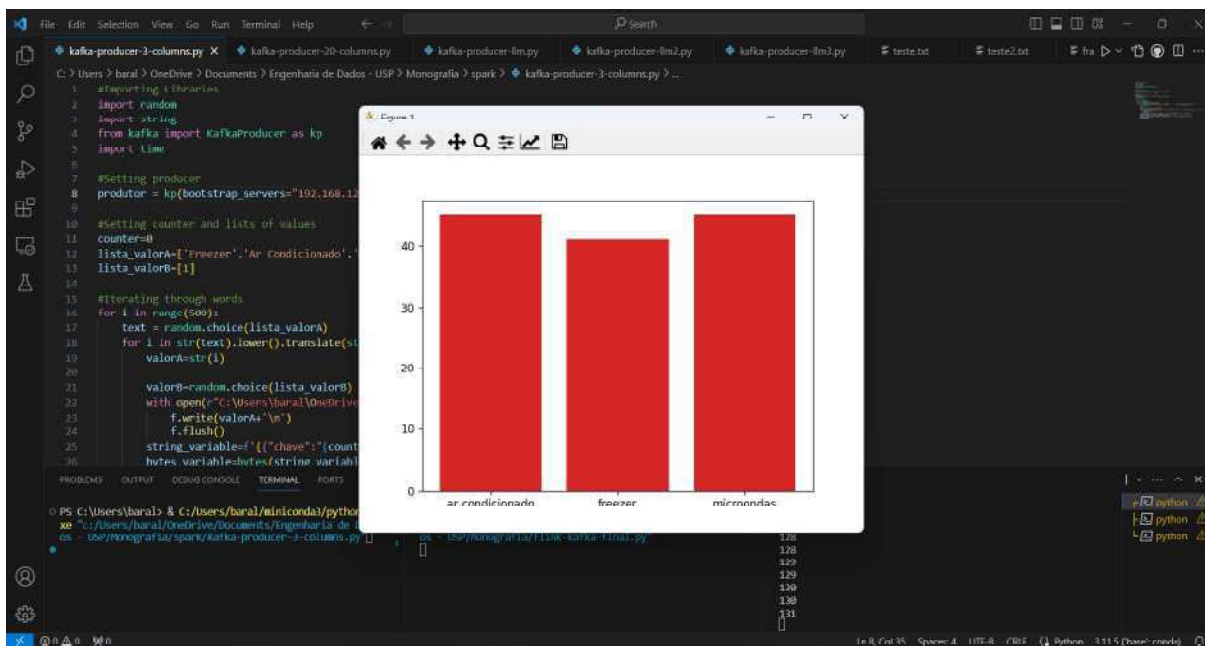
```
1 #Importing Libraries
2 import random
3 import string
4 from kafka import KafkaProducer as kp
5 import time
6
7 #Setting producer
8 producer = kp(bootstrap_servers="192.168.126.126:9092")
9
10 #Setting counter and lists of values
11 counter=0
12 lista_valores=['Freezer','Ar condicionado','Microondas']
13 lista_valores=[1]
14
15 #Iterating through words
16 for i in range(500):
17     text = random.choice(lista_valores)
18     for j in str(text).lower().translate(str.maketrans('', '', string.punctuation)).split('\n'):
19         valorA=str(i)
20
21         valorB=random.choice(lista_valores)
22         with open("C:/Users/baral/OneDrive/Documents/Engenharia de Dados - USP/Monografia/teste.txt", 'a') as f:
23             f.write(valorA+'\n')
24             f.flush()
25
26         string_variable=f"{i} have {counter} {valorA} {valorA} {valorA} {valorA} {valorA}"
27         bytes_variable=bytes(string_variable.encode('utf-8'))
```

Terminal output:

```
PS C:\Users\baral> & C:/Users/baral/miniconda3/python.exe "c:/Users/baral/OneDrive/Documents/Engenharia de Dados - USP/Monografia/spark/kafka-producer-3-columns.py"
0
1
1
2
2
3
```

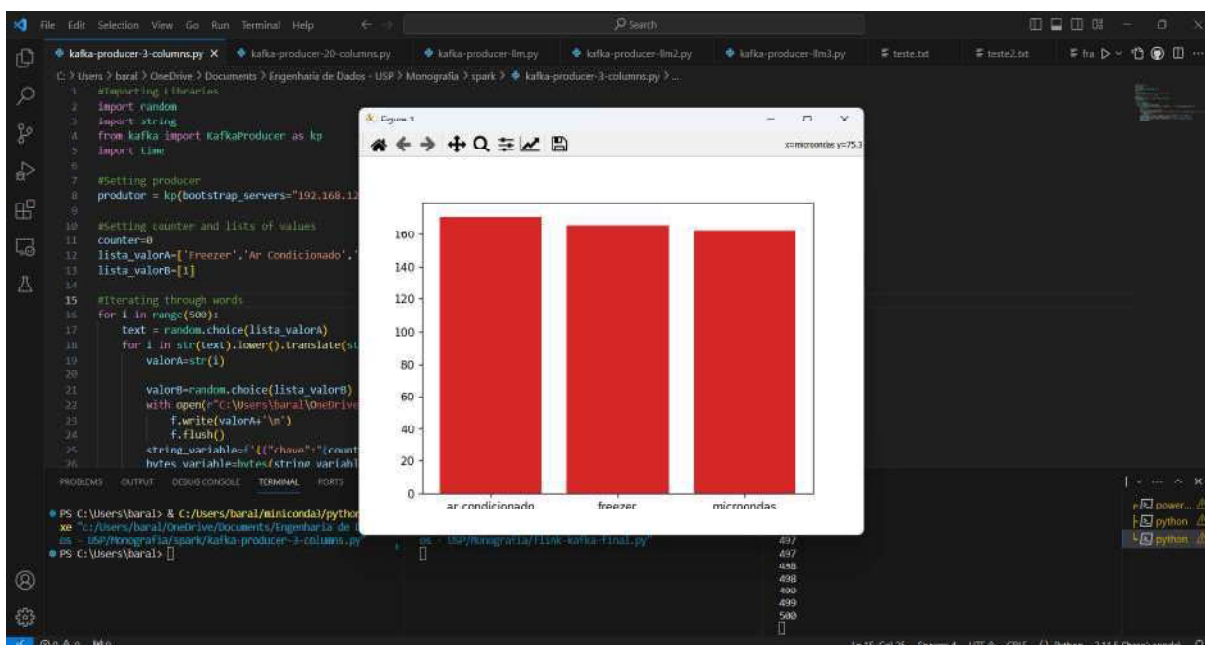
Fonte: criada pelo autor

Figura B-17 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-18 - Visualização Final



Fonte: criada pelo autor

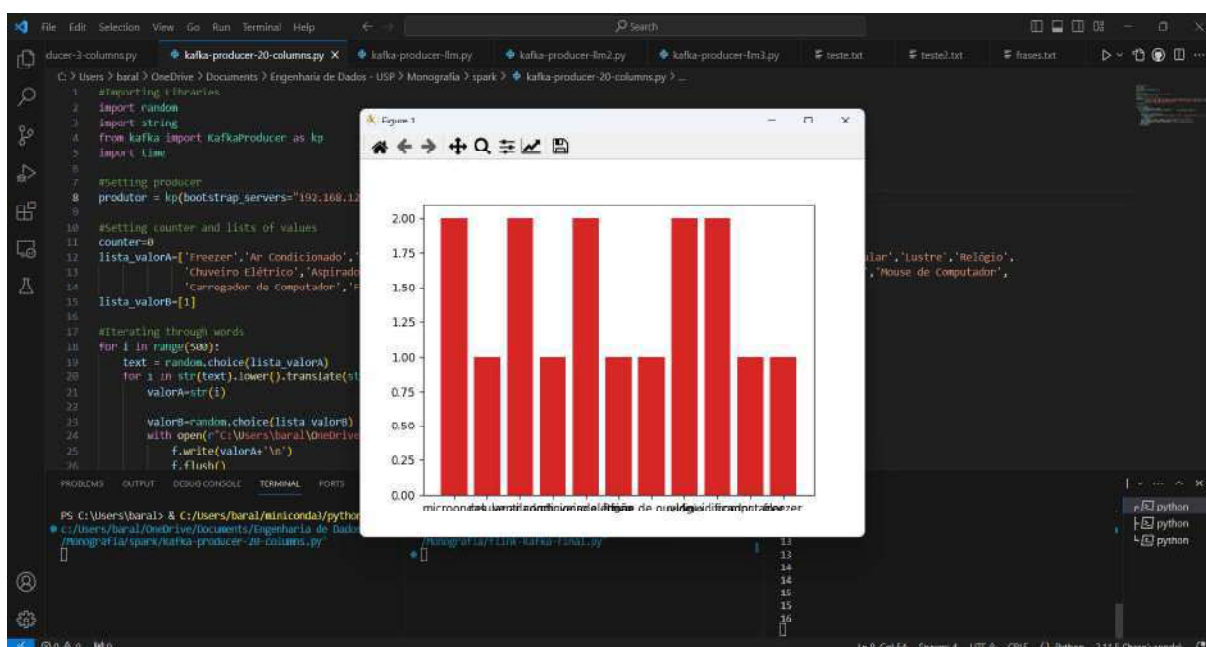
Tabela B-6 - Amostras por Palavra

Gráfico de 3 Colunas	Contagem Total de Amostras	Contagem Freezer	Contagem Ar Condicionado	Contagem Microondas
	500	166	171	163

Fonte: criada pelo autor

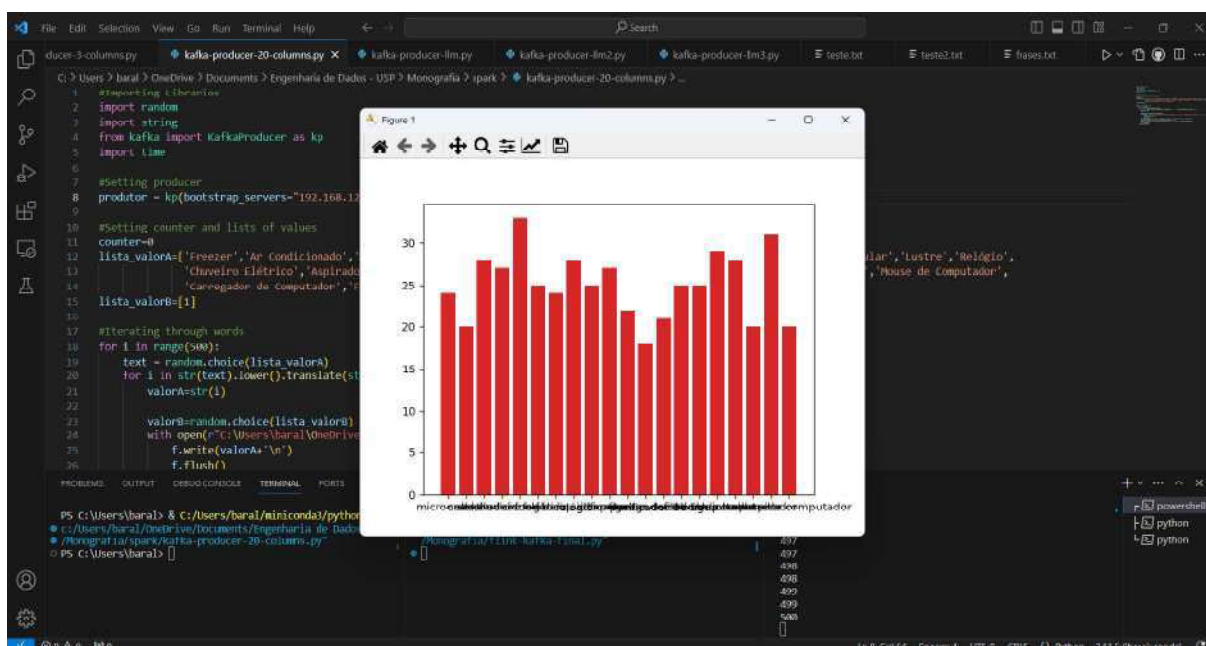
B.1.1.2.2 VINTE COLUNAS/PRODUTOS SEM TRATAMENTO (500 EXEMPLOS)

Figura B-19 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-20 - Visualização Final



Fonte: criada pelo autor

Tabela B-7 - Contagem Total de Amostras com 500 Exemplos

Contagem Ar Condicionado	Contagem Aspirador	Contagem Carregador de Celular	Contagem Carregador de Computador	Contagem Celular
27	31	18	25	20

Contagem Chuveiro Elétrico	Contagem Computador	Contagem Filtro de Linha	Contagem Fogão	Contagem Fone de Ouvido
33	27	25	25	24

Contagem Freezer	Contagem Liquidificador	Contagem Lustre	Contagem Microondas	Contagem Mouse de Computador
22	25	20	24	20

Contagem Purificador de Água	Contagem Relógio	Contagem Roteador	Contagem Teclado de Computador	Contagem Ventilador
21	28	28	29	28

Fonte: criada pelo autor

B.1.1.2.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF E SEM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-21 - Execução dos Scripts

```

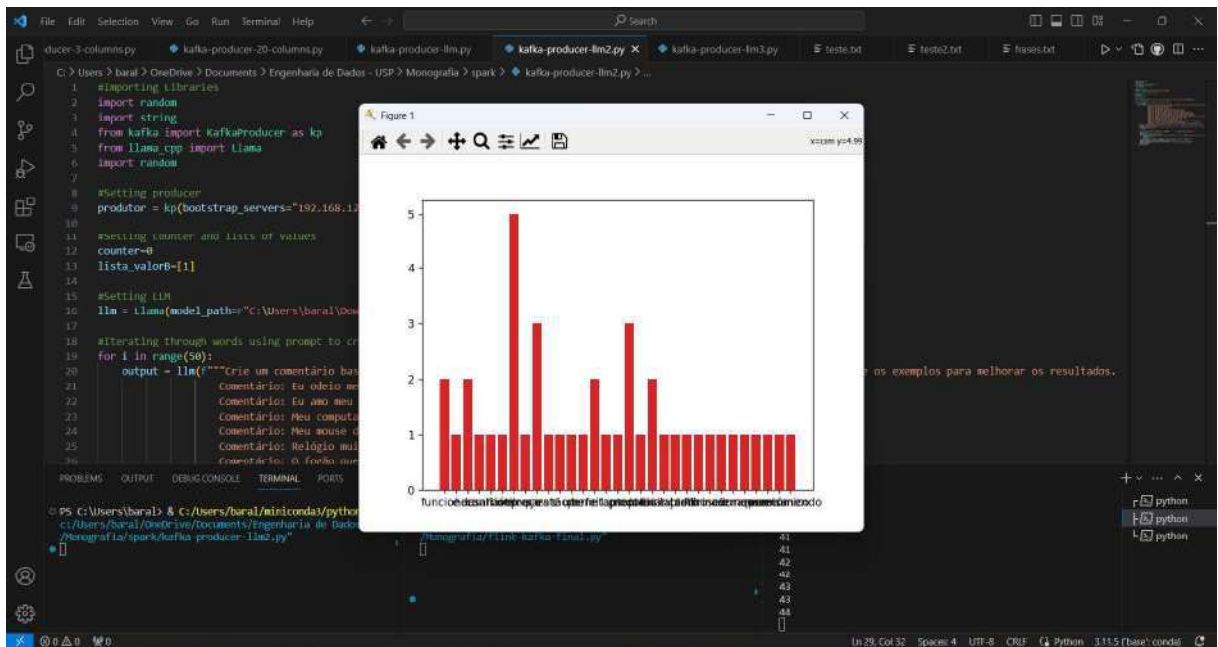
1 #Importing libraries
2 import random
3 import string
4 from kafka import KafkaProducer as kp
5 from llama_cpp import Llama
6 import random
7
8 #Setting producer
9 produtor = kp(bootstrap_servers="192.168.128.128:9092")
10
11 #Setting counter and lists of values
12 counter=0
13 lista_valores=[1]
14
15 #Setting LLM
16 lla = Llama(model_path="C:\Users\baral\Downloads\llama-2-7b-chat-gguf", verbose=False)
17
18 #Iterating through words using prompt to create comments
19 for i in range(50):
20     output = lla(prompt="Crie um comentário bastante curto positivo ou negativo sobre um produto que voce comprou recentemente. Use os exemplos para melhorar os resultados.
21     Comentário: Eu odeio meu liquidificador.
22     Comentário: Eu amo meu microondas.
23     Comentário: Meu computador é incrível.
24     Comentário: Meu mouse de computador não funciona muito bem.
25     Comentário: Relógio muito bom, bem melhor que o meu antigo.
26     Comentário: O fralda que comprei tá está vive de fatur. Não funciona não.")

```

Fonte: criada pelo autor

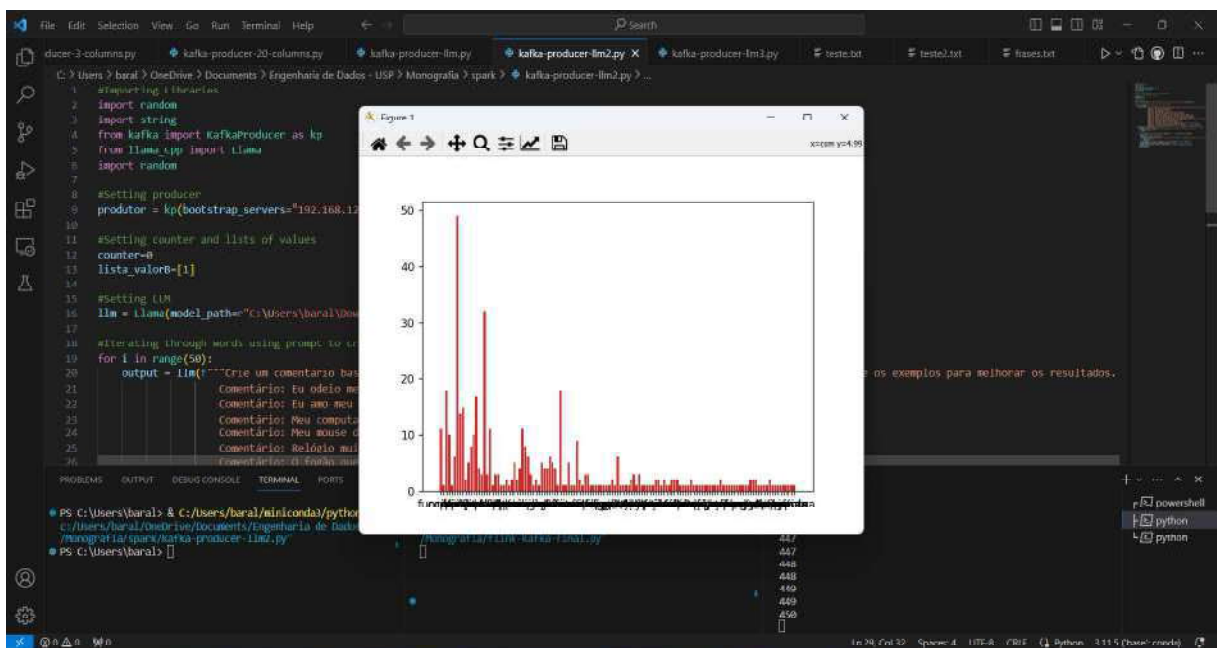
B.1.1.2.4 THEBLOKE/LLAMA-2-13B-CHAT-GGUF E SEM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-23 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-24 - Visualização Final



Fonte: criada pelo autor

Tabela B-9 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	450	450

Fonte: criada pelo autor

B.1.1.2.5 THEBLOKE/ZEPHYR-7B-BETA-GGUF E SEM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-25 - Execução dos *Scripts*

The image shows a code editor with several Python files and a terminal window. The main file is `kafka-producer-llm3.py`, which contains the following code:

```

1 #!python3
2 import random
3 import string
4 from kafka import KafkaProducer as kp
5 from llama_cpp import Llama
6 import random
7
8 #setting producer
9 producer = kp(bootstrap_servers="192.166.126.128:9092")
10
11 #Setting counter and lists of values
12 counter=0
13 lista_valores=[1]
14
15 #setting llm
16 llm = Llama(model_path="C:/Users/baral/Downloads/zephyr_7b_beta_gguf.gguf", verbose=False)
17
18 #iterating through words using prompt to create comments
19 for i in range(50):
20     output = llm(prompt="Crie um comentário bastante curto positivo ou negativo sobre um produto que voce comprou recentemente. Use os exemplos para melhorar os resultados.")
21     print(output)
22     comentario = output
23     print(comentario)
24     comentario = "Meu mouse de computador não funciona muito bem."
25     print(comentario)
26     comentario = "Relógio muito bom, sem melhor que o meu antigo."
27     print(comentario)
28     comentario = "O fone que cancela o ruído está com defeito, não recomendo."
29     print(comentario)

```

The terminal window shows the execution of the scripts. The first terminal window shows the execution of `kafka-producer-llm3.py`, which outputs the following comments:

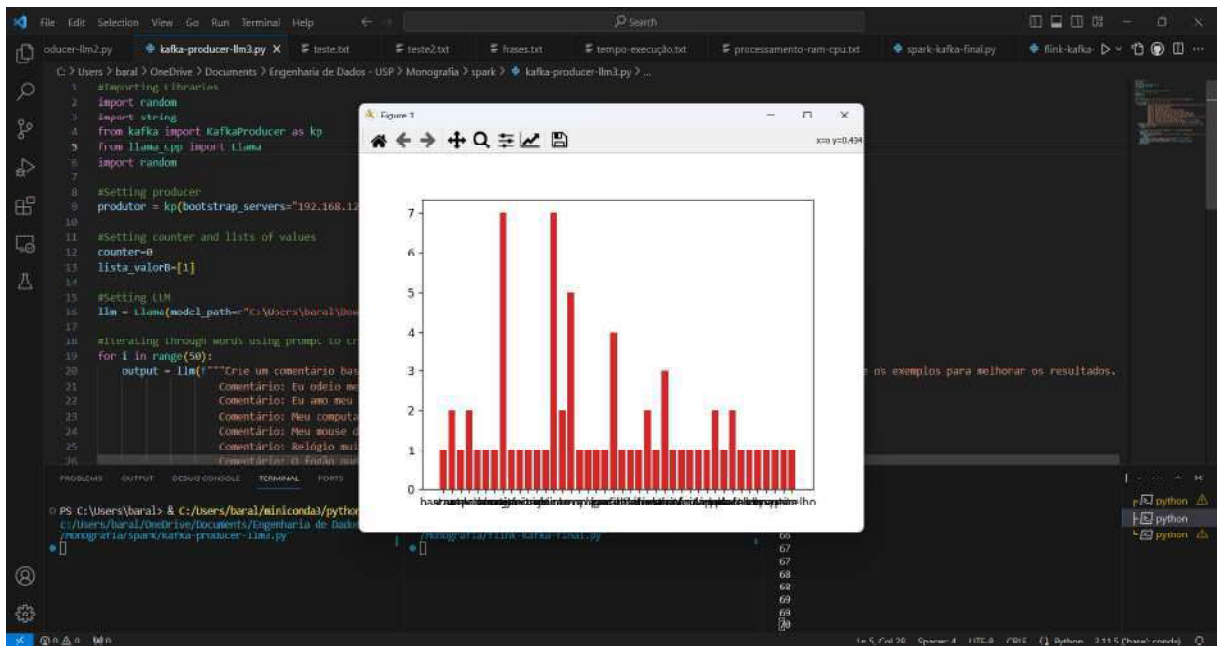
```

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

```

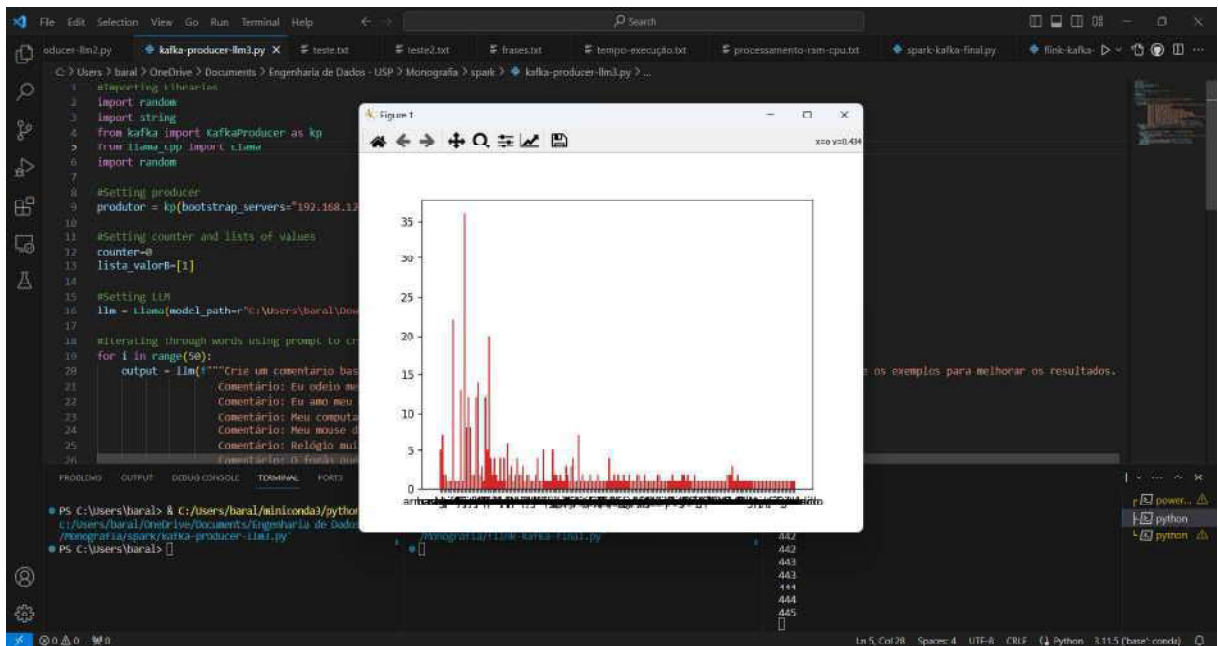
Fonte: criada pelo autor

Figura B-26 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-27 - Visualização Final



Fonte: criada pelo autor

Tabela B-10 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	445	445

Fonte: criada pelo autor

B.1.2 COM TRATAMENTO

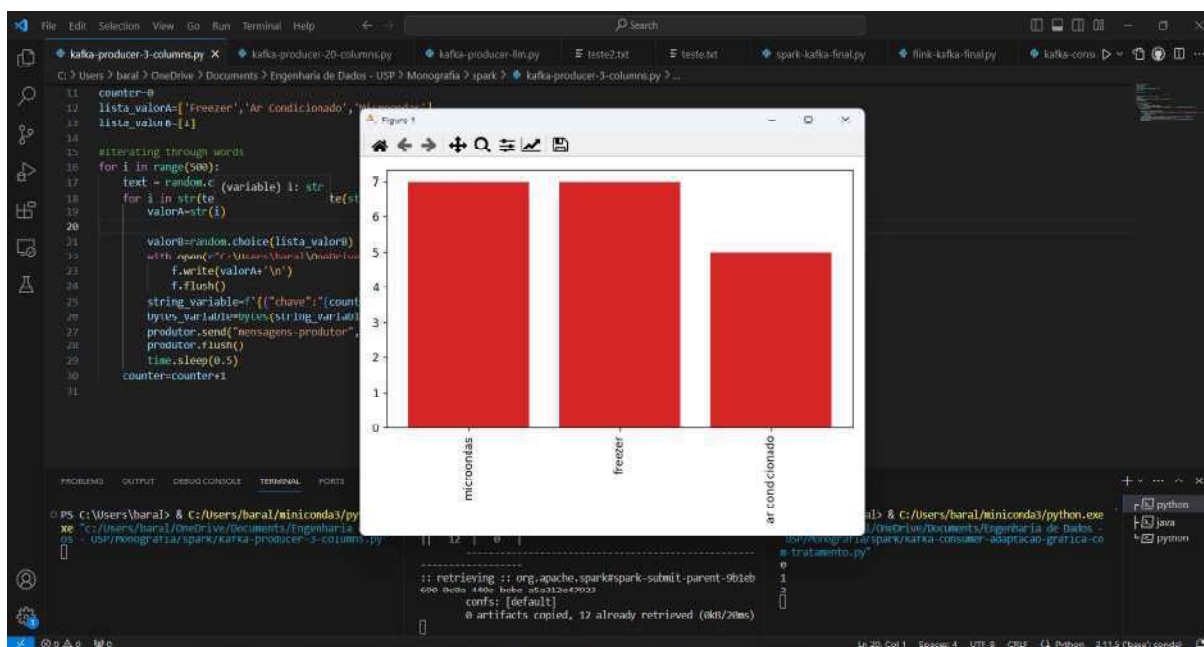
Visualizações com tratamento para PySpark e PyFlink.

B.1.2.1 PYSPARK

Cinco abordagens utilizando PySpark.

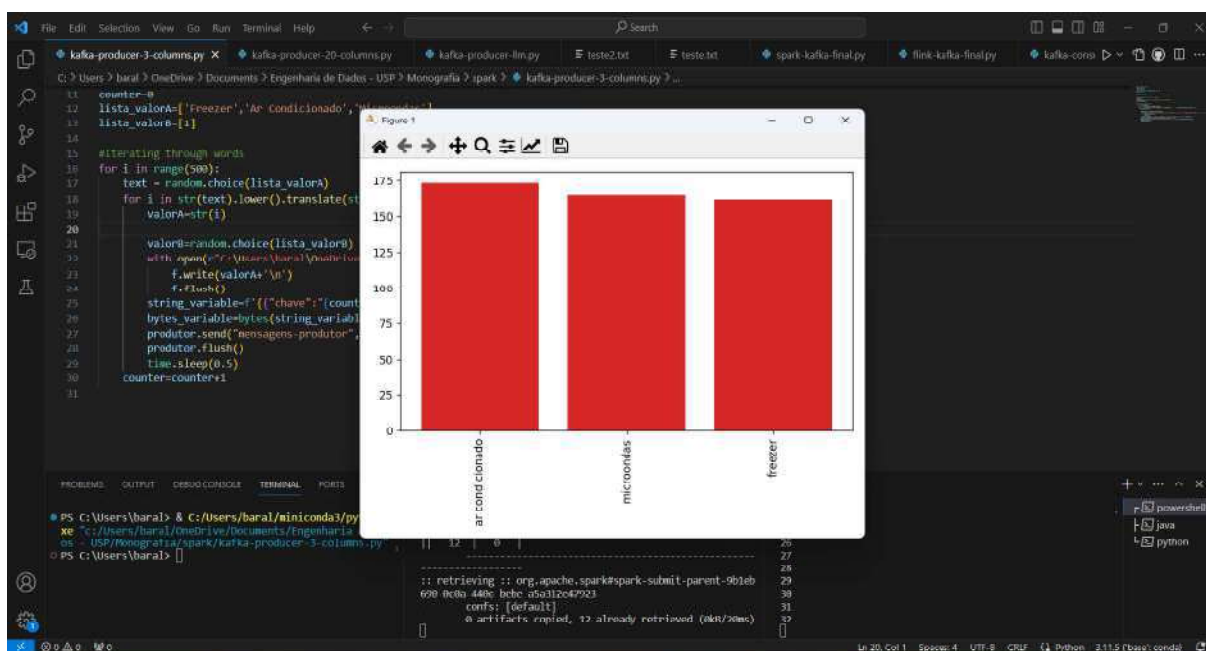
B.1.2.1.1 TRÊS COLUNAS/PRODUTOS E COM TRATAMENTO (500 EXEMPLOS)

Figura B-28 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-29 - Visualização Final



Fonte: criada pelo autor

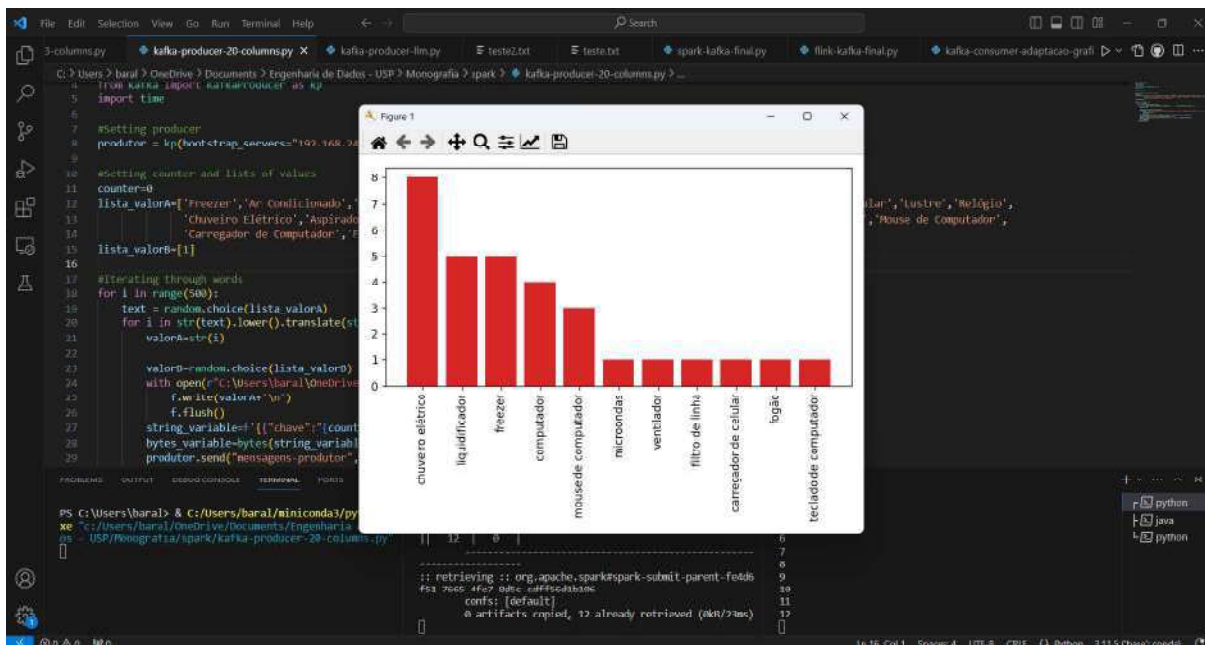
Tabela B-11 - Amostras por Palavra

Gráfico de 3 Colunas	Contagem Total de Amostras	Contagem Freezer	Contagem Ar Condicionado	Contagem Microondas
	500	162	173	165

Fonte: criada pelo autor

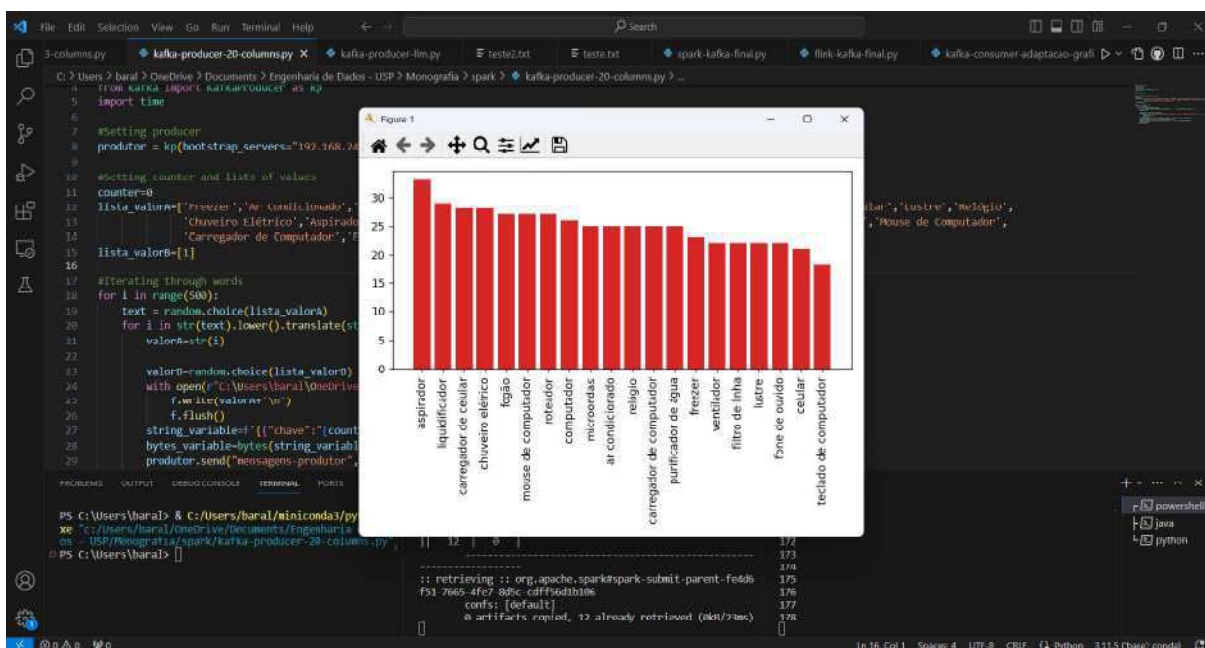
B.1.2.1.2 VINTE COLUNAS/PRODUTOS E COM TRATAMENTO (500 EXEMPLOS)

Figura B-30 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-31 - Visualização Final



Fonte: criada pelo autor

Tabela B-12 - Contagem Total de Amostras com 500 Exemplos

Contagem Ar Condicionado	Contagem Aspirador	Contagem Carregador de Celular	Contagem Carregador de Computador	Contagem Celular
25	33	28	25	21

Contagem Chuveiro Elétrico	Contagem Computador	Contagem Filtro de Linha	Contagem Fogão	Contagem Fone de Ouvido
28	26	22	27	22

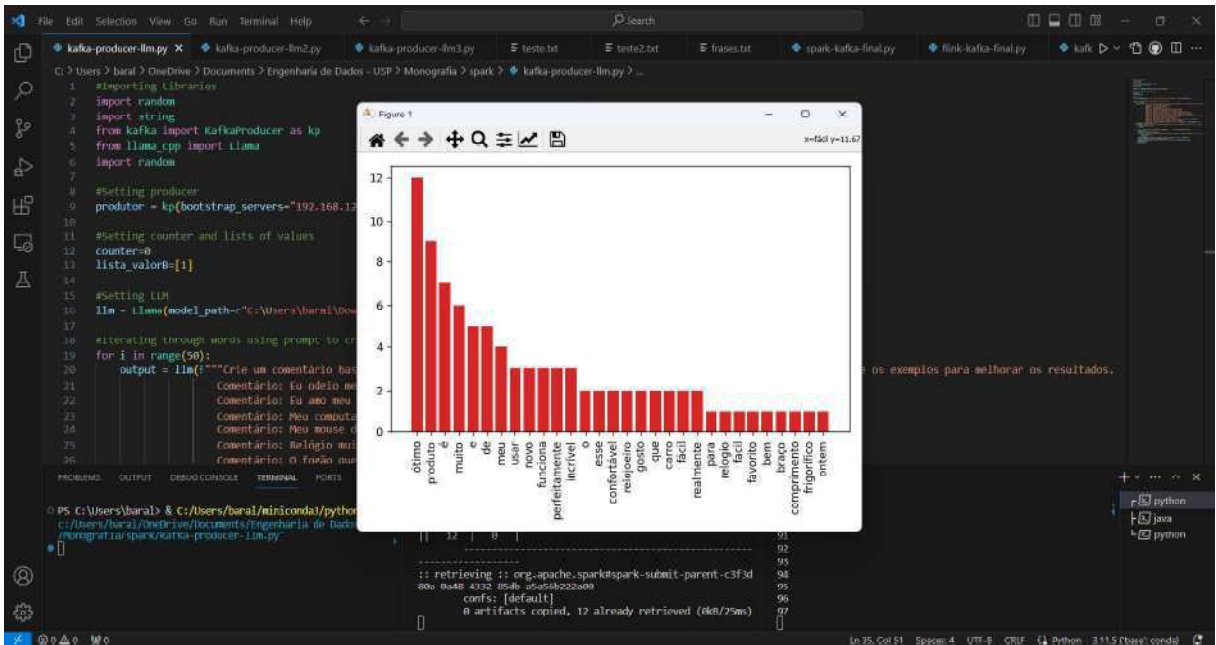
Contagem Freezer	Contagem Liquidificador	Contagem Lustre	Contagem Microondas	Contagem Mouse de Computador
23	29	22	25	27

Contagem Purificador de Água	Contagem Relógio	Contagem Roteador	Contagem Teclado de Computador	Contagem Ventilador
25	25	27	18	22

Fonte: criada pelo autor

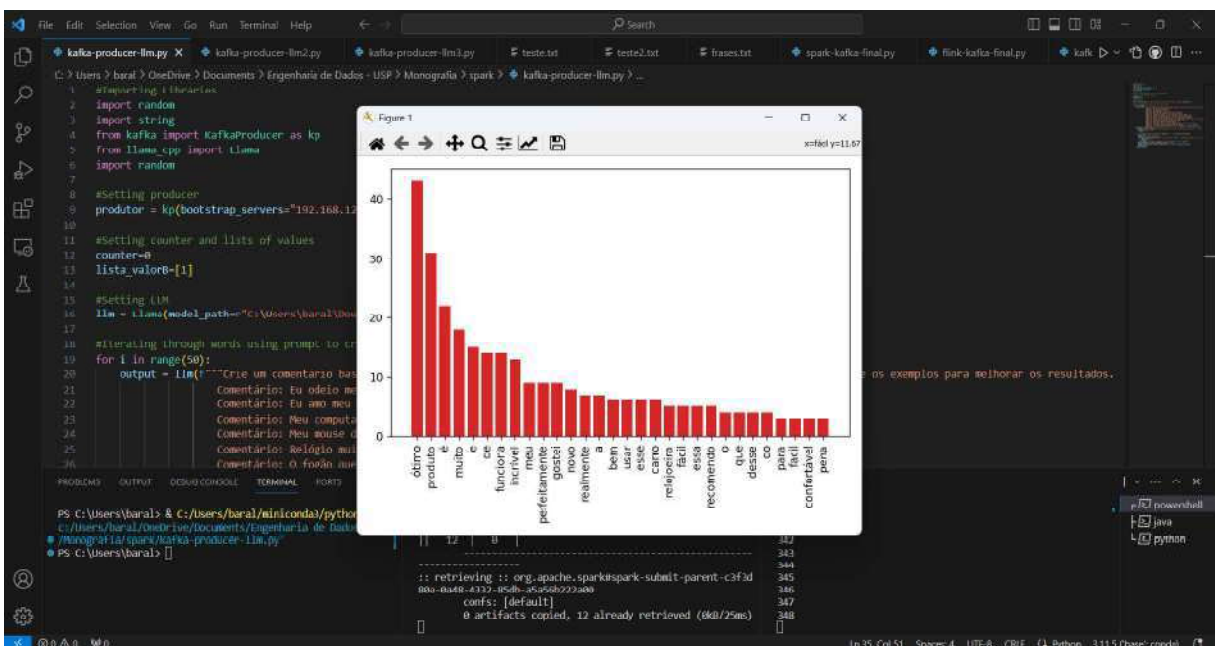
B.1.2.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF COM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-32 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-33 - Visualização Final



Fonte: criada pelo autor

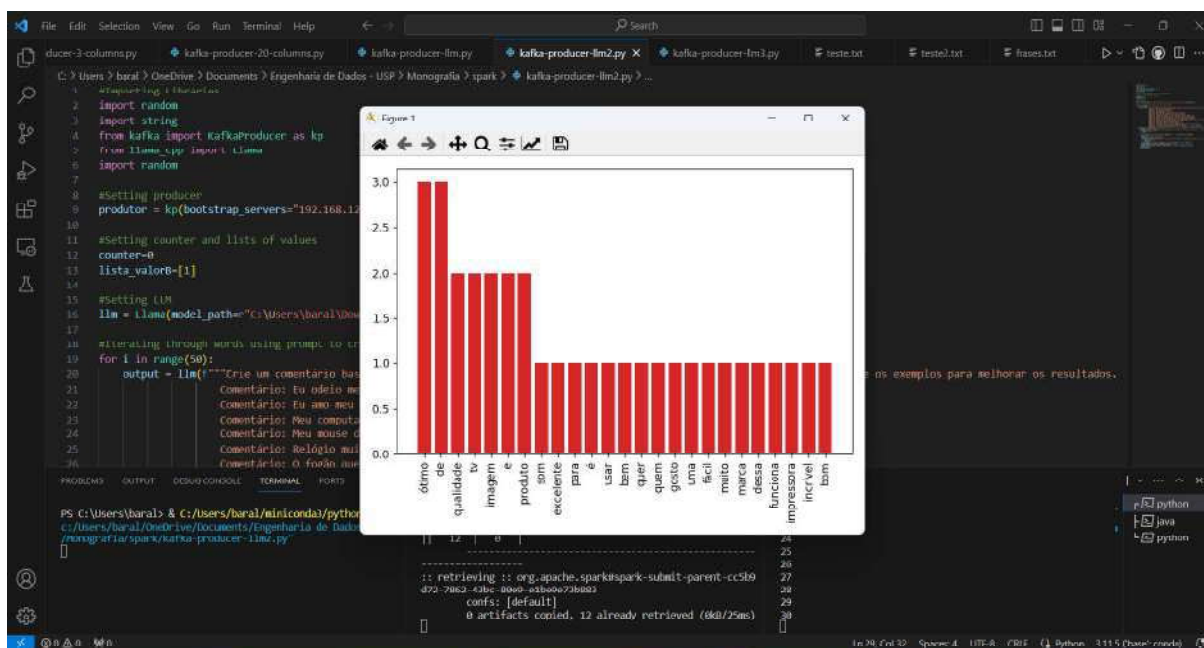
Tabela B-13 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	406	406

Fonte: criada pelo autor

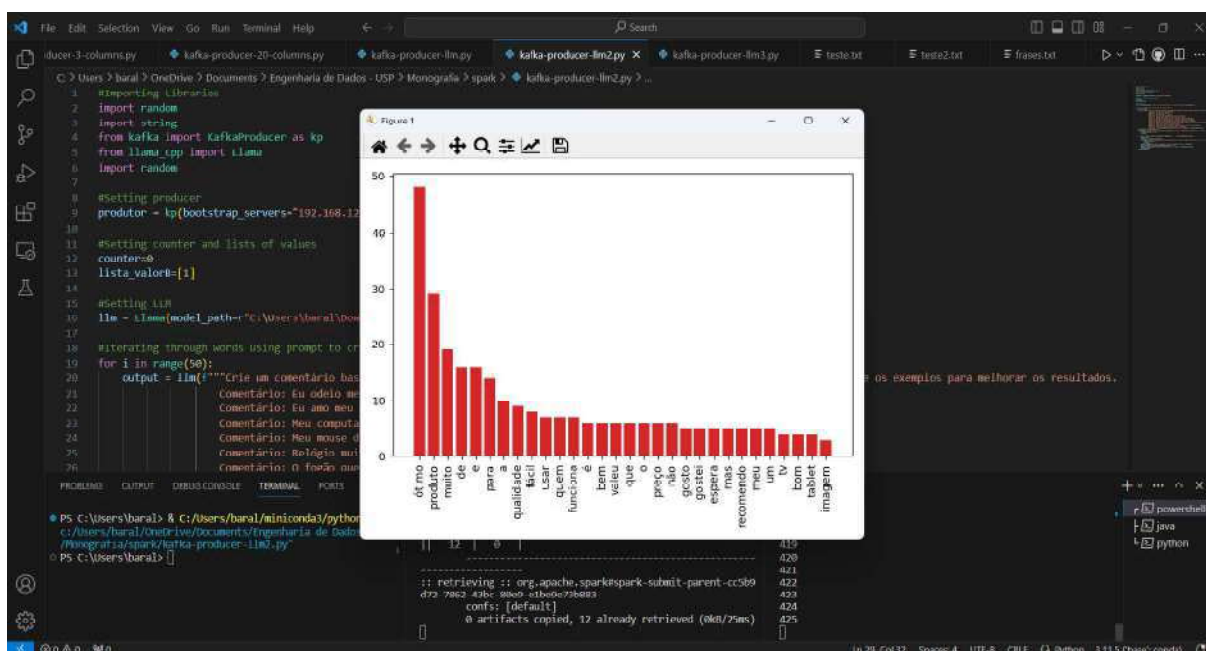
B.1.2.1.4 THEBLOKE/LLAMA-2-13B-CHAT-GGUF COM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-34 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-35 - Visualização Final



Fonte: criada pelo autor

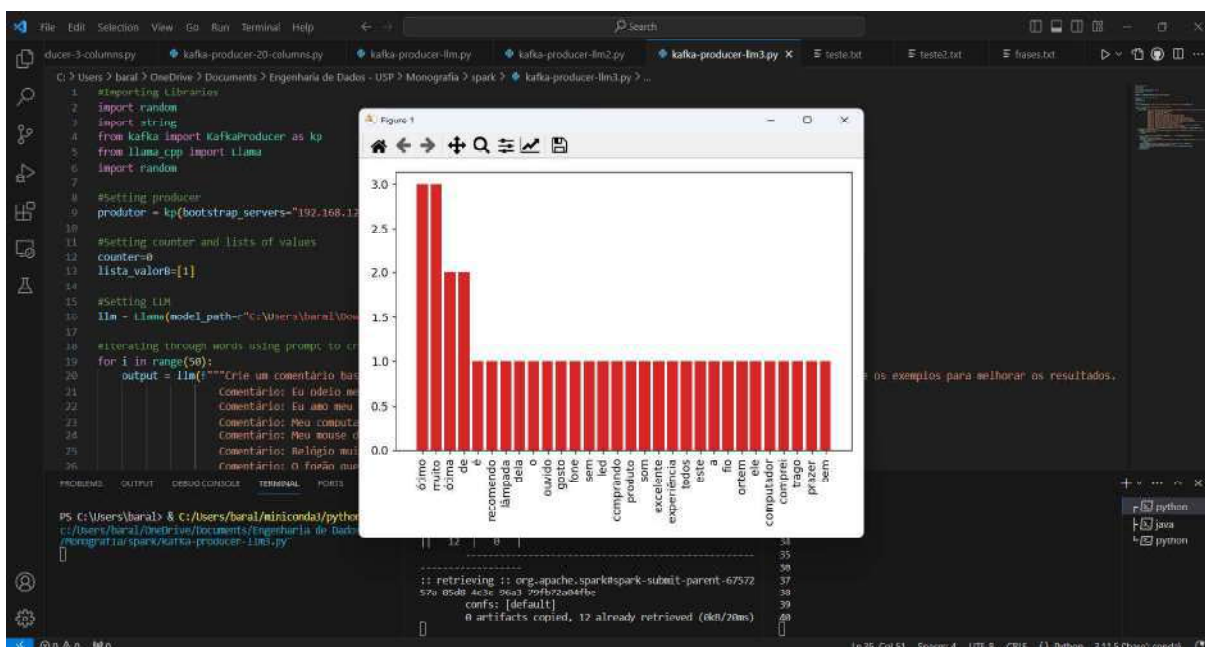
Tabela B-14 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	449	449

Fonte: criada pelo autor

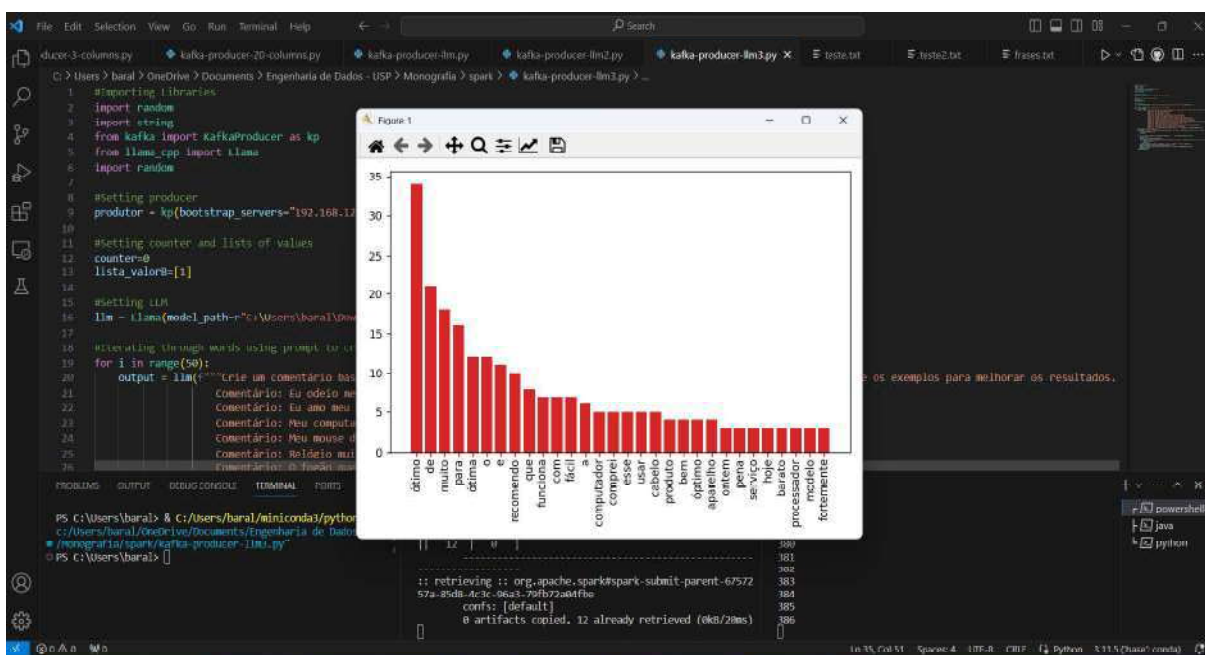
B.1.2.1.5 THEBLOKE/ZEPHYR-7B-BETA-GGUFLLM COM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-36 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-37 - Visualização Final



Fonte: criada pelo autor

Tabela B-15 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	410	410

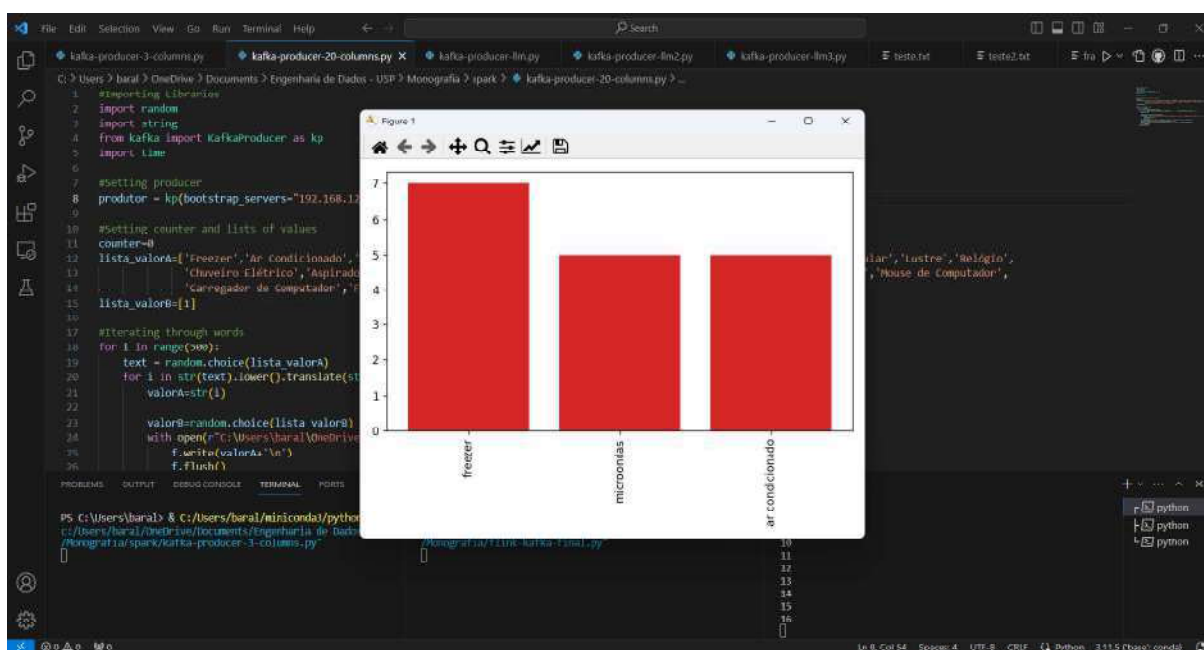
Fonte: criada pelo autor

B.1.2.2 PYFLINK

Cinco abordagens utilizando PyFlink.

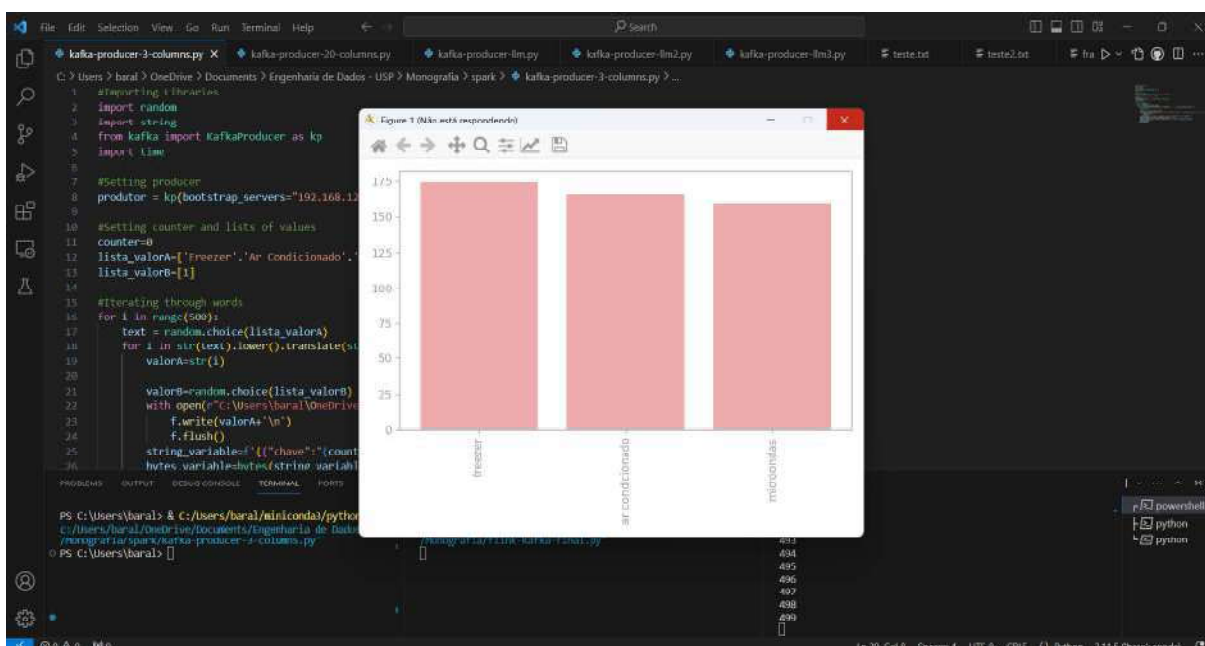
B.1.2.2.1 TRÊS COLUNAS/PRODUTOS COM TRATAMENTO (500 EXEMPLOS)

Figura B-38 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-39 - Visualização Final



Fonte: criada pelo autor

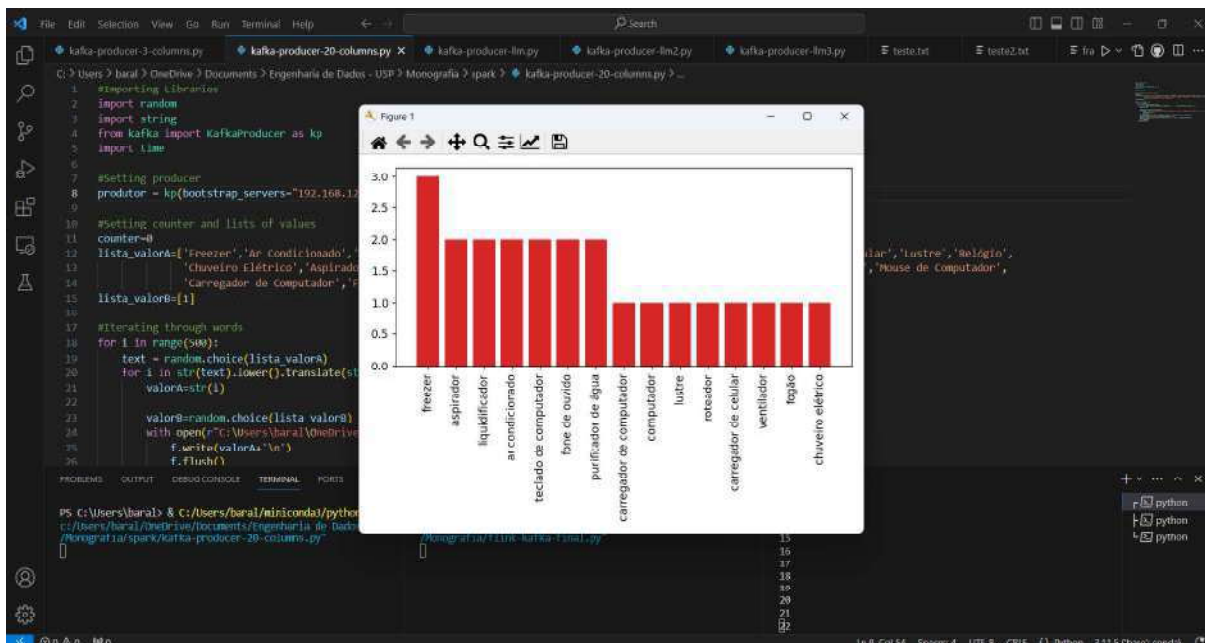
Tabela B-16 - Amostras por Palavra

Gráfico de 3 Colunas	Contagem Total de Amostras	Contagem Freezer	Contagem Ar Condicionado	Contagem Microondas
	500	174	166	160

Fonte: criada pelo autor

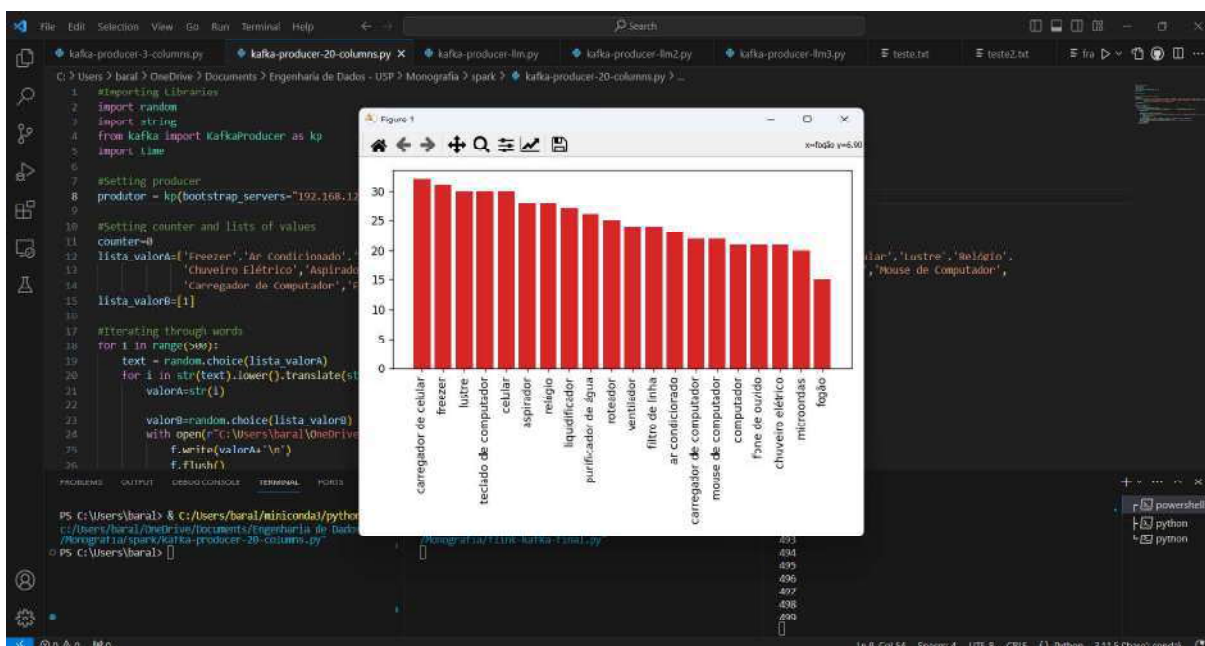
B.1.2.2.2 VINTE COLUNAS/PRODUTOS E COM TRATAMENTO (500 EXEMPLOS)

Figura B-40 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-41 - Visualização Final



Fonte: criada pelo autor

Tabela B-17 - Contagem Total de Amostras com 500 Exemplos

Contagem Ar Condicionado	Contagem Aspirador	Contagem Carregador de Celular	Contagem Carregador de Computador	Contagem Celular
23	28	32	22	30

Contagem Chuveiro Elétrico	Contagem Computador	Contagem Filtro de Linha	Contagem Fogão	Contagem Fone de Ouvido
21	21	24	15	21

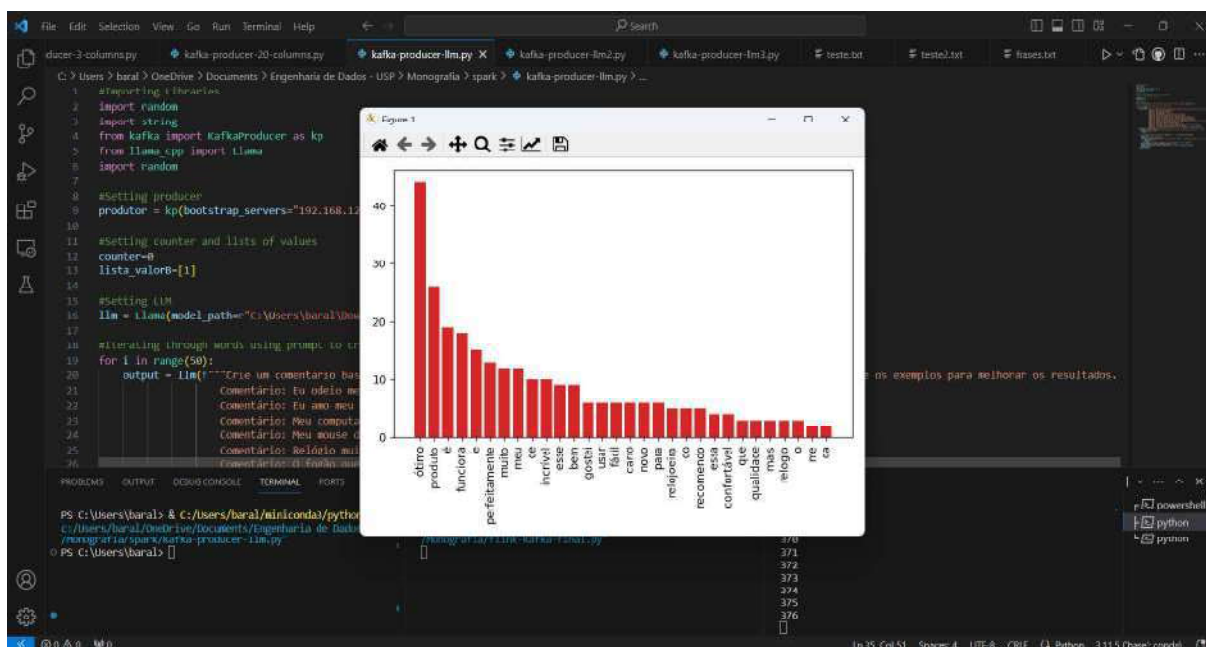
Contagem Freezer	Contagem Liquidificador	Contagem Lustre	Contagem Microondas	Contagem Mouse de Computador
31	27	30	20	22

Contagem Purificador de Água	Contagem Relógio	Contagem Roteador	Contagem Teclado de Computador	Contagem Ventilador
26	28	25	30	24

Fonte: criada pelo autor

B.1.2.2.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF E COM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-42 - Visualização Final



Fonte: criada pelo autor

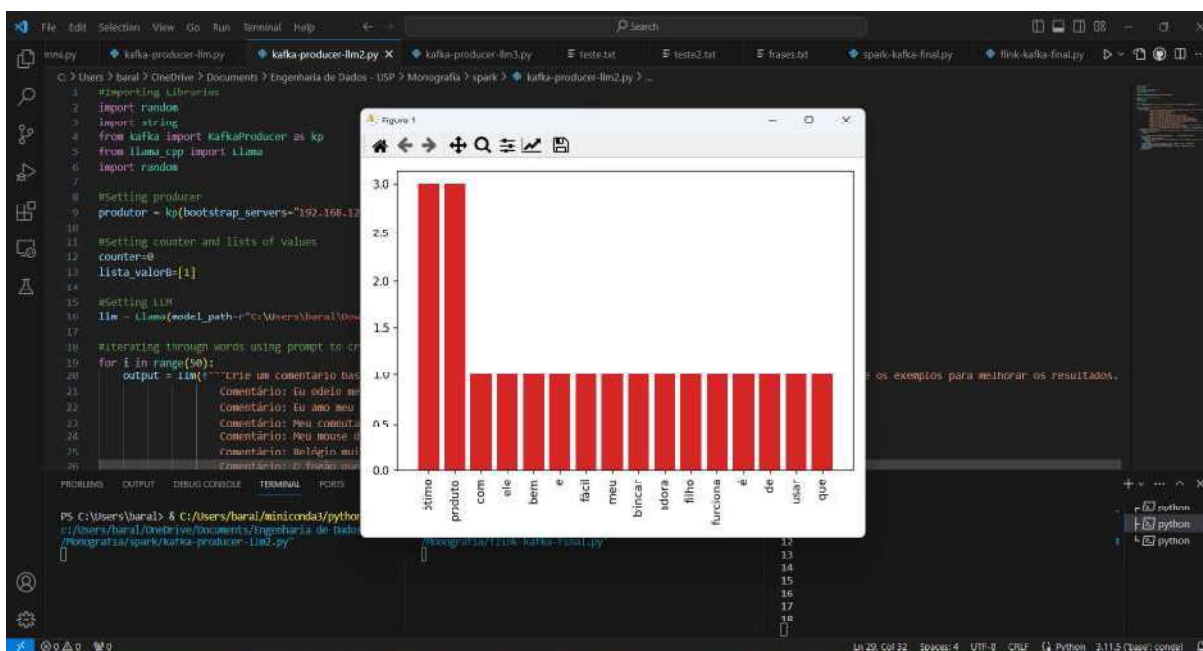
Tabela B-18 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	377	377

Fonte: criada pelo autor

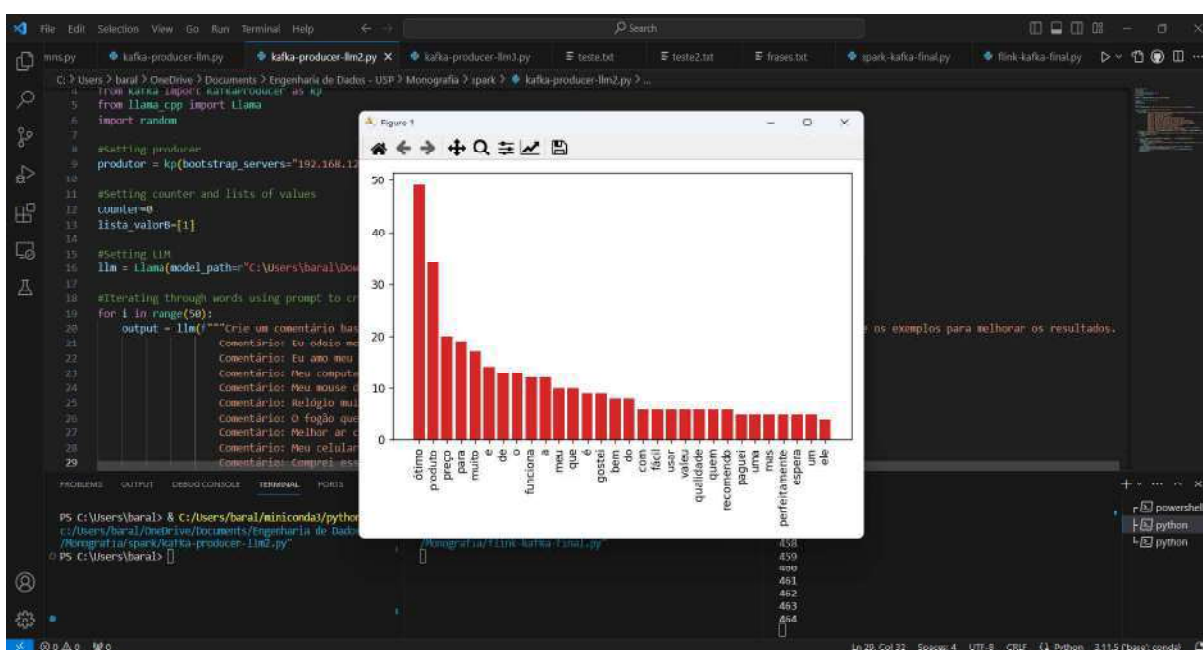
B.1.2.2.4 THEBLOKE/LLAMA-2-13B-CHAT-GGUF E COM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-43 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-44 - Visualização Final



Fonte: criada pelo autor

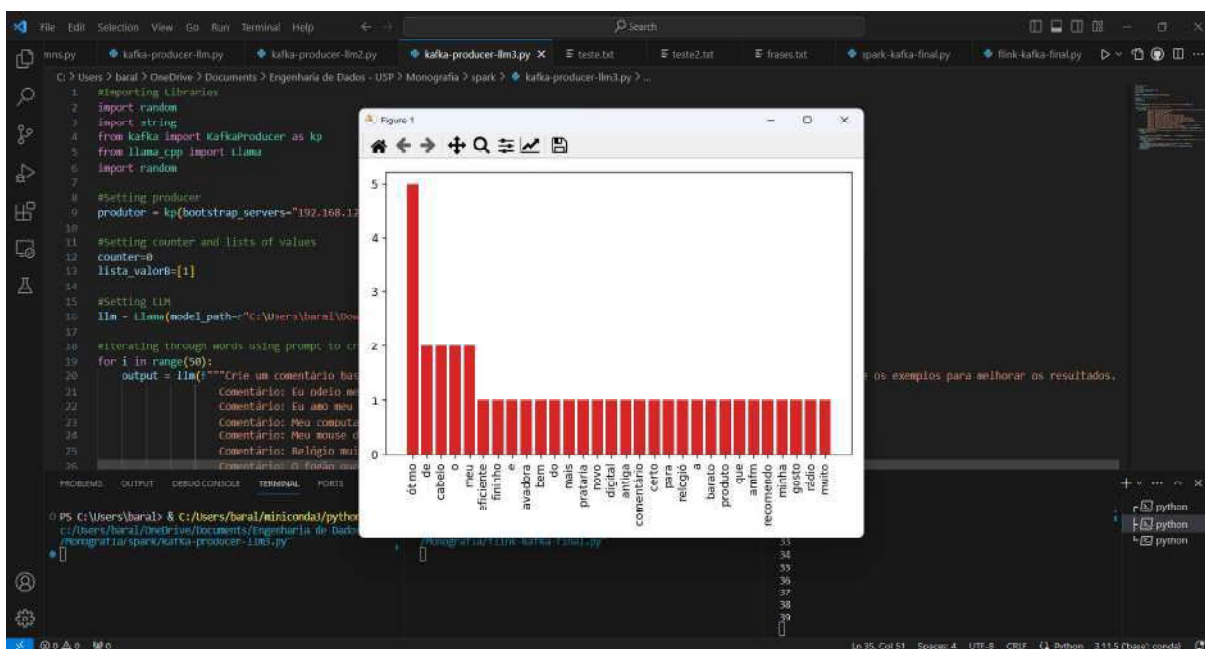
Tabela B-19 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	465	465

Fonte: criada pelo autor

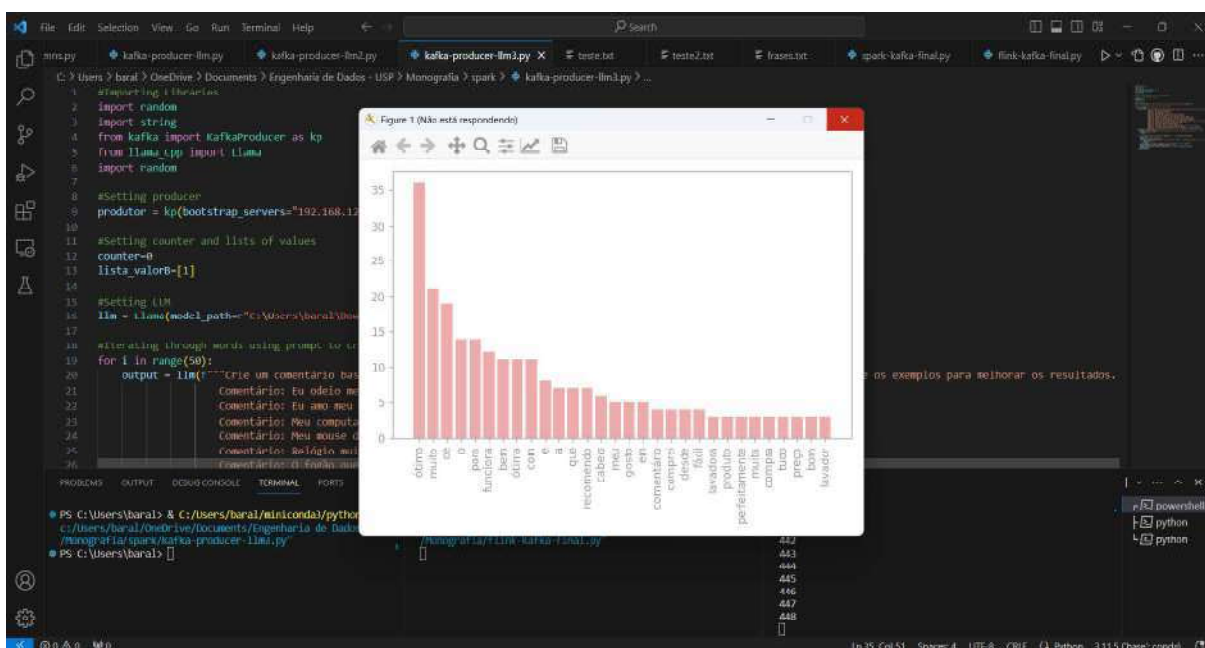
B.1.2.2.5 THEBLOKE/ZEPHYR-7B-BETA-GGUF E COM TRATAMENTO (50 GERAÇÕES DE TEXTO)

Figura B-45 - Primeiras Atualizações Gráficas



Fonte: criada pelo autor

Figura B-46 - Visualização Final



Fonte: criada pelo autor

Tabela B-20 - Contagem Total de Amostras com 50 Iterações de Comentários

Contagem Total de Iterações de Comentários	Palavras Totais de Entrada	Palavras Totais de Saída
50	449	449

Fonte: criada pelo autor

B.2 RESULTADO DO TEMPO DE ATUALIZAÇÃO

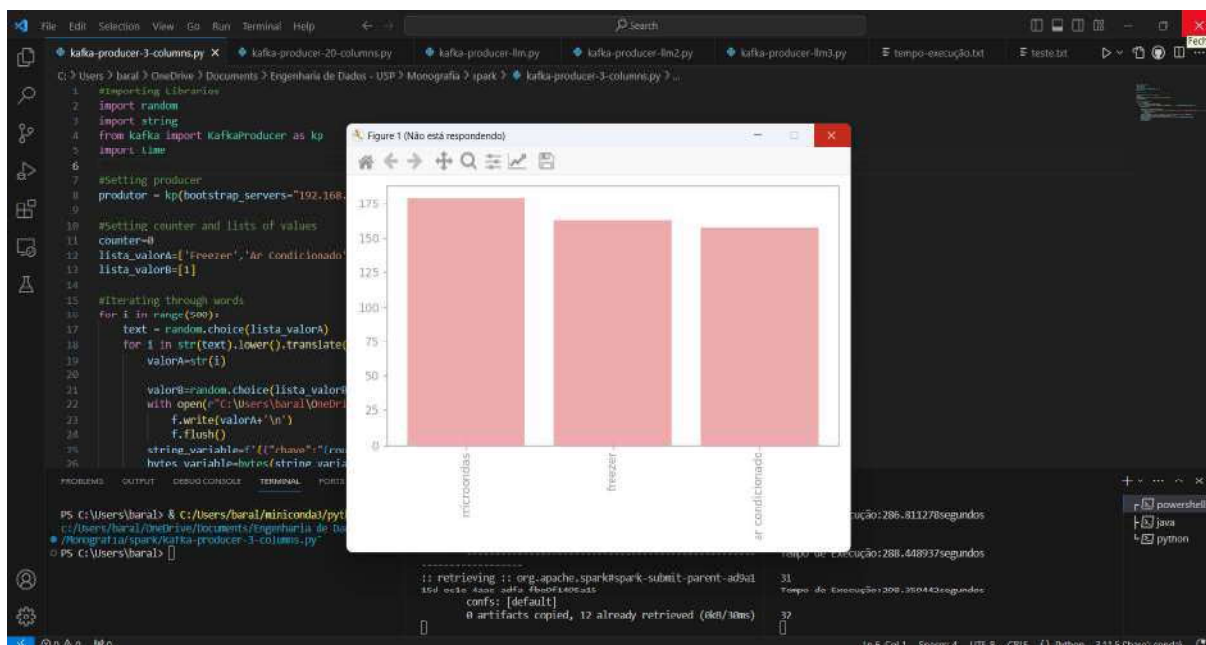
Resultados do tempo de atualização gráfica para PySpark e PyFlink com o uso de tratamento gráfico.

B.2.1.1 PYSPARK

Cinco abordagens utilizando PySpark.

B.2.1.1.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura B-47 - Visualização Final



Fonte: criada pelo autor

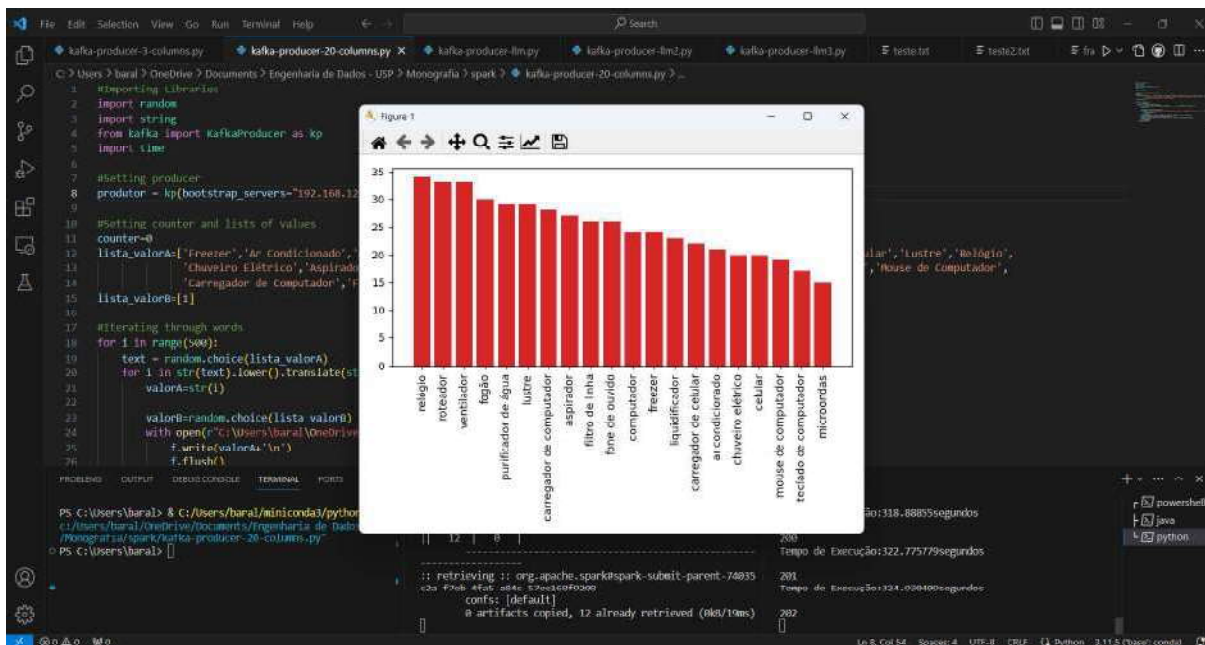
Tabela B-21 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
9,041	7,875

Fonte: criada pelo autor

B.2.1.1.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura B-48 - Visualização Final



Fonte: criada pelo autor

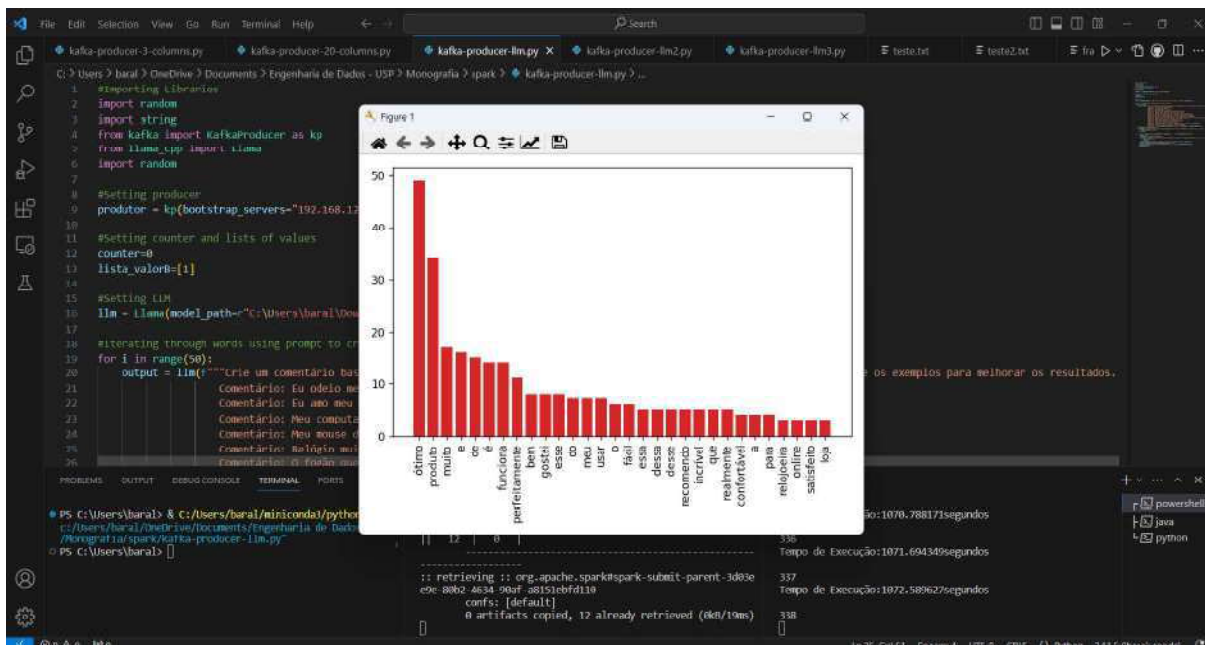
Tabela B-22 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
1,601	3,072

Fonte: criada pelo autor

B.2.1.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-49 - Visualização Final



Fonte: criada pelo autor

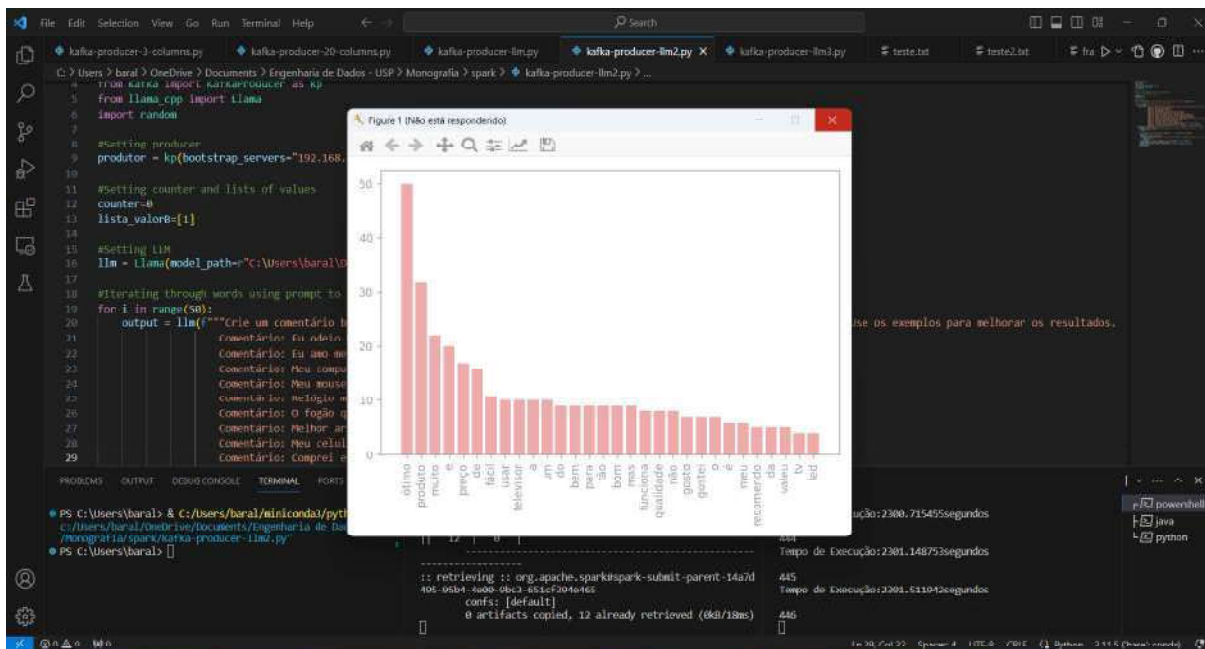
Tabela B-23 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
3,164	7,459

Fonte: criada pelo autor

B.2.1.1.4 THEBLOKE/LLAMA-2-13B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-50 - Visualização Final



Fonte: criada pelo autor

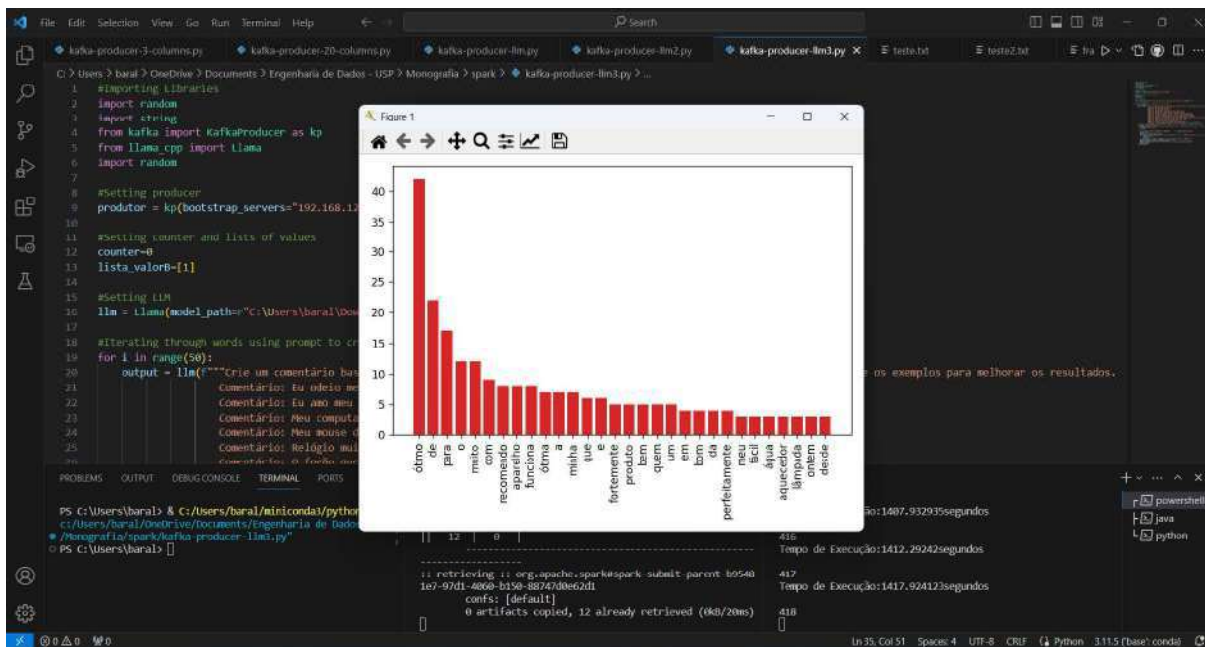
Tabela B-24 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
4,753	9,728

Fonte: criada pelo autor

B.2.1.1.5 THEBLOKE/ZEPHYR-7B-BETA-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-51 - Visualização Final



Fonte: criada pelo autor

Tabela B-25 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
3,384	7,566

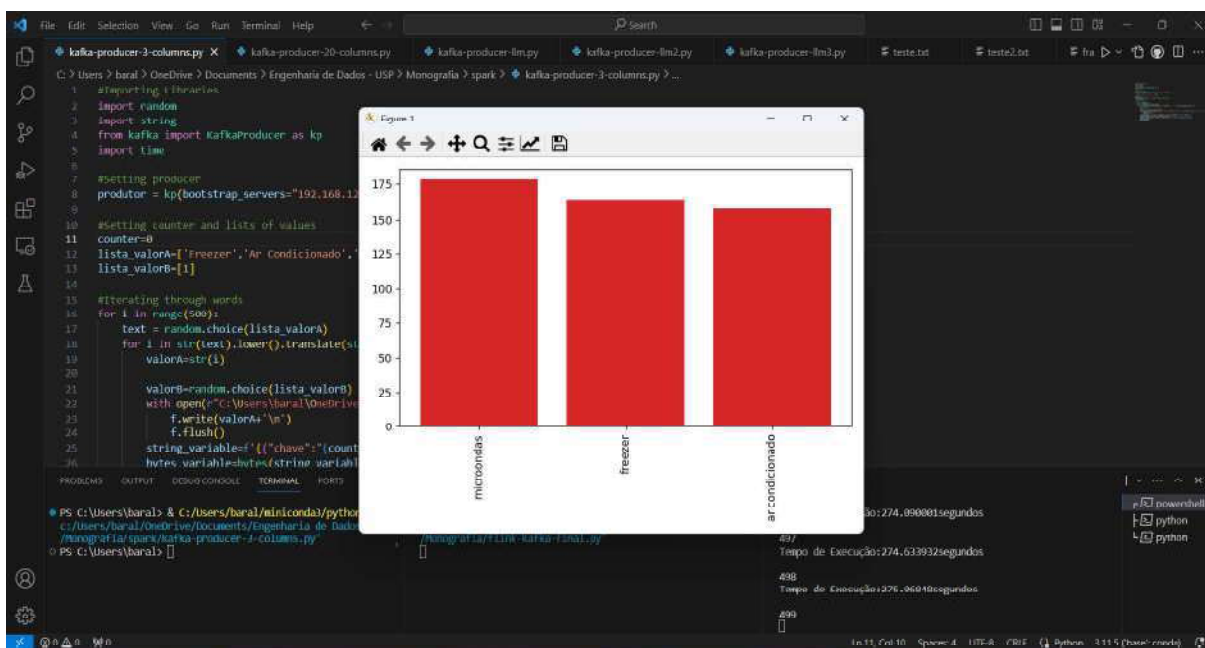
Fonte: criada pelo autor

B.2.1.2 PYFLINK

Cinco abordagens utilizando PyFlink.

B.2.1.2.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura B-52 - Visualização Final



Fonte: criada pelo autor

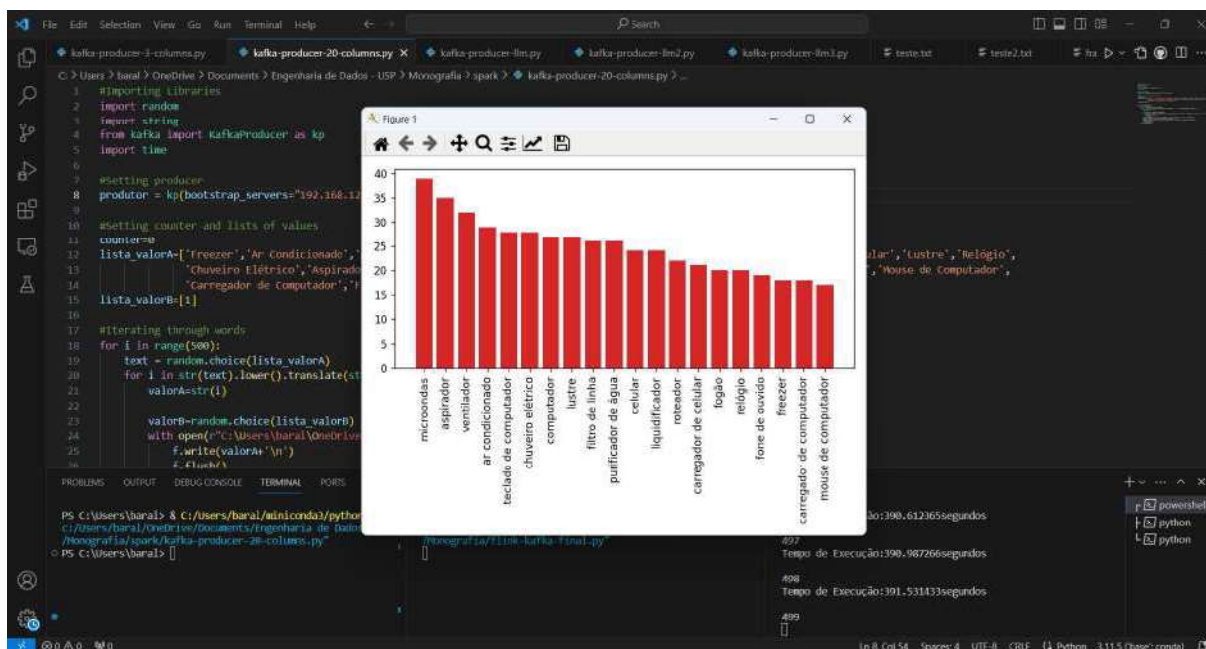
Tabela B-26 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
0,550	1,492

Fonte: criada pelo autor

B.2.1.2.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura B-53 - Visualização Final



Fonte: criada pelo autor

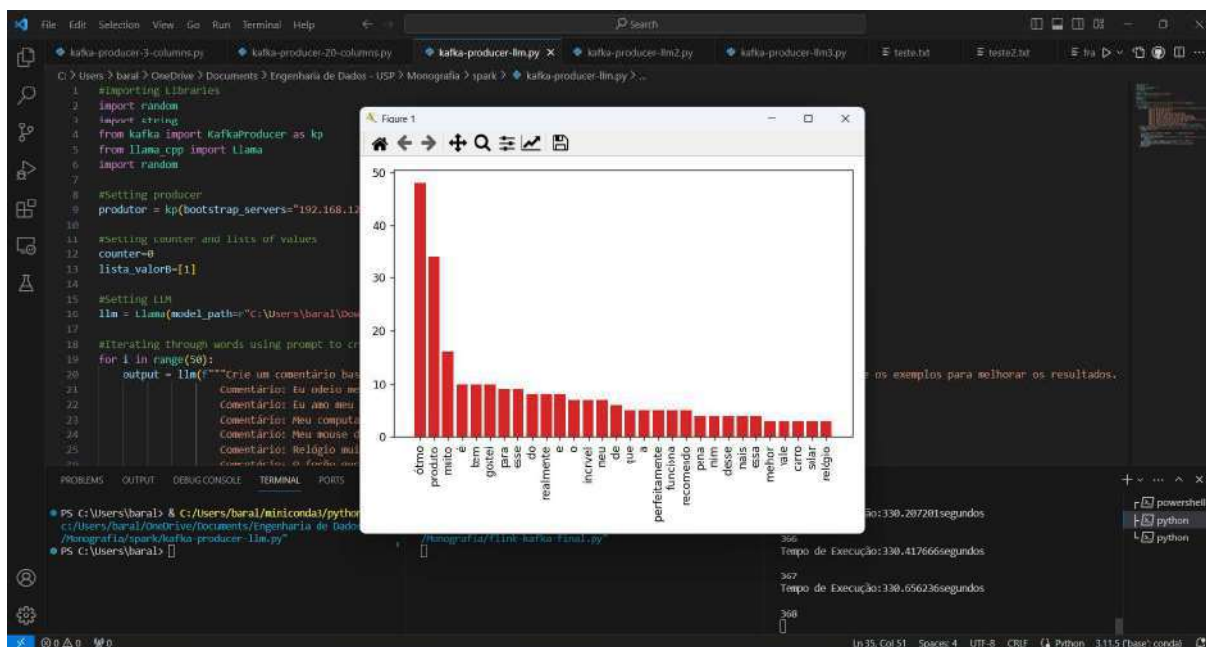
Tabela B-27 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
0,783	6,382

Fonte: criada pelo autor

B.2.1.2.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-54 - Visualização Final



Fonte: criada pelo autor

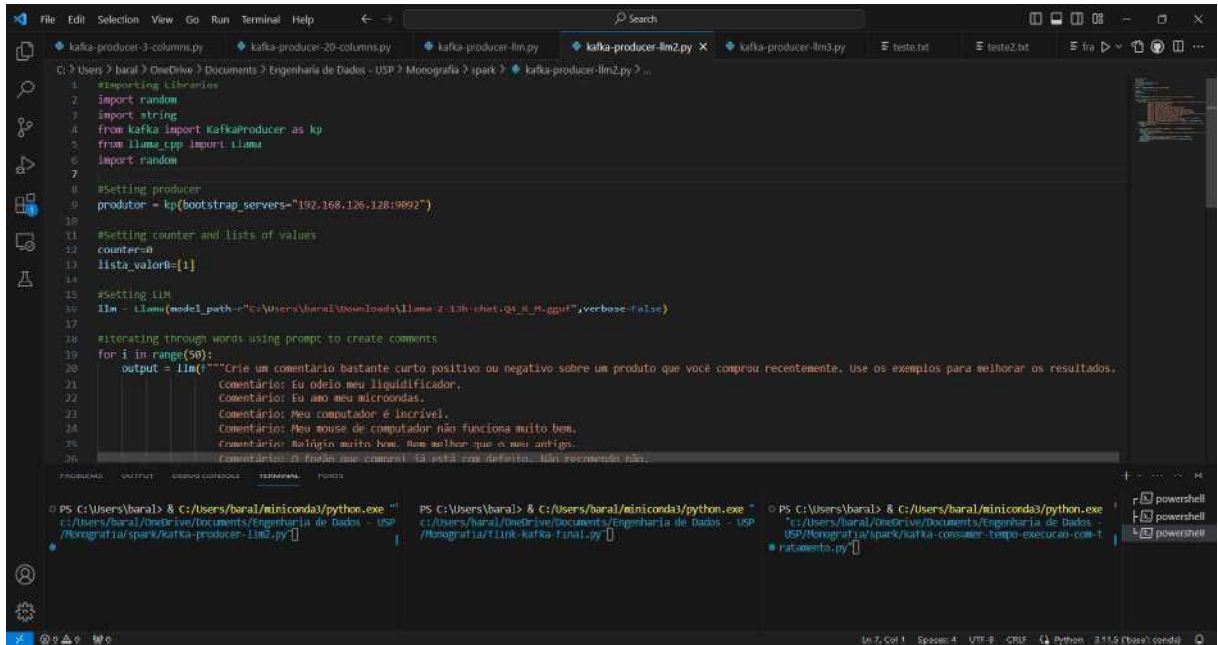
Tabela B-28 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
0,896	4,754

Fonte: criada pelo autor

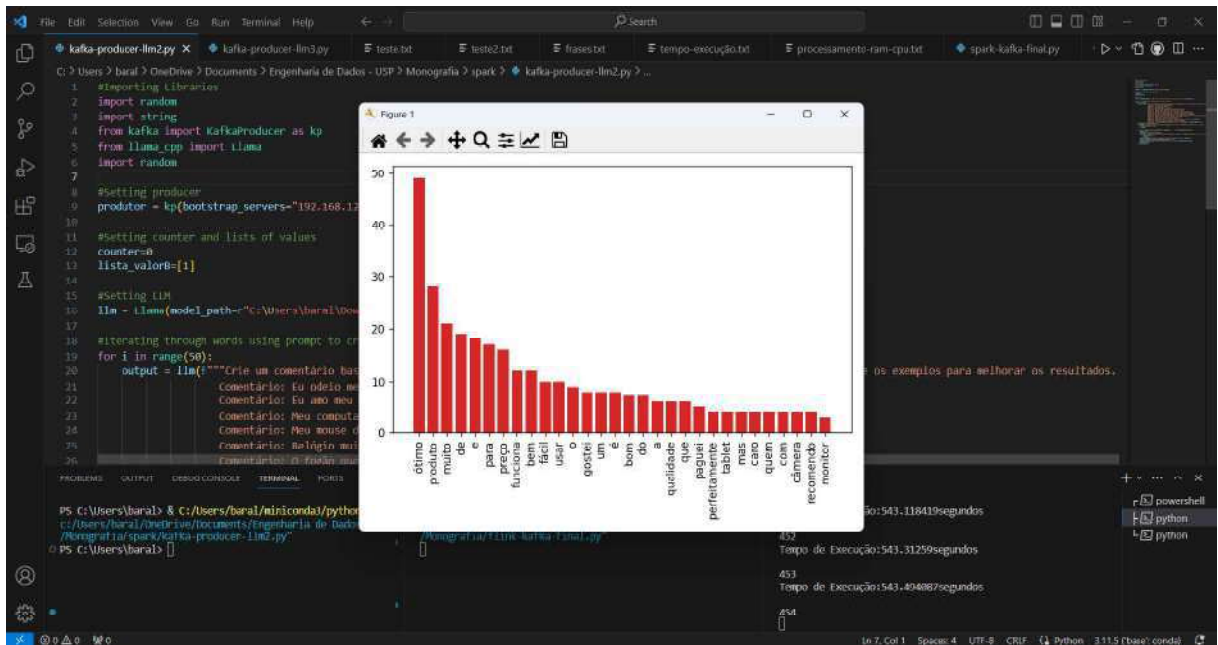
B.2.1.2.4 THEBLOKE/LLAMA-2-13B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-55 - Execução dos Scripts



Fonte: criada pelo autor

Figura B-56 - Visualização Final



Fonte: criada pelo autor

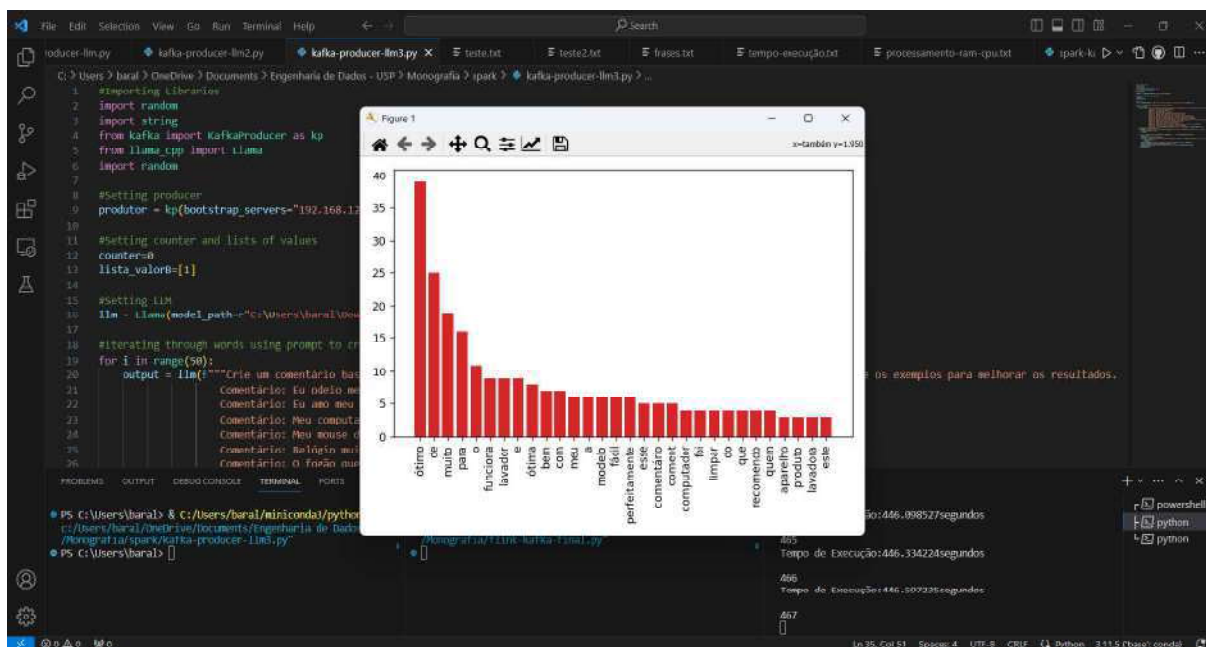
Tabela B-29 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
1,194	5,250

Fonte: criada pelo autor

B.2.1.2.5 THEBLOKE/ZEPHYR-7B-BETA-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-57 - Visualização Final



Fonte: criada pelo autor

Tabela B-30 - Média e Desvio Padrão do Tempo de Atualização em Segundos

MÉDIA	DESV. PADRÃO
0,954	3,797

Fonte: criada pelo autor

B.2.2 CONSUMO DE CPU E MEMÓRIA RAM

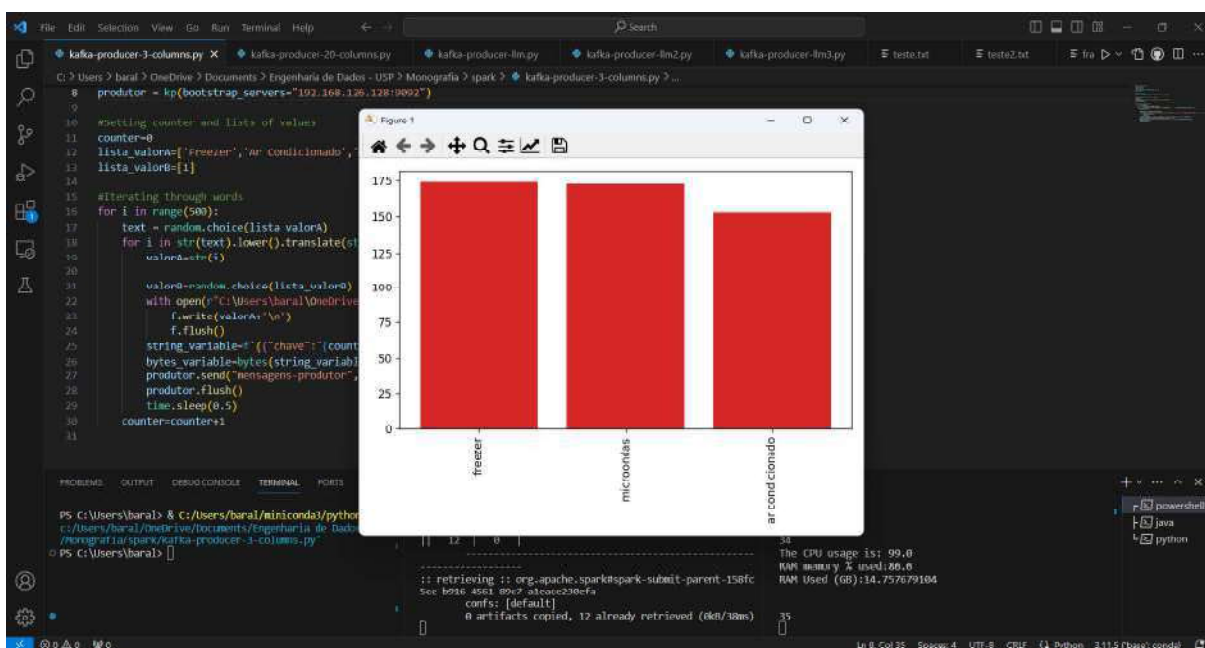
Resultados do consumo de CPU e memória RAM com respectivos percentuais para as tecnologias PySpark e PyFlink.

B.2.2.1 PYSPARK

Cinco abordagens utilizando PySpark.

B.2.2.1.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura B-58 - Visualização Final



Fonte: criada pelo autor

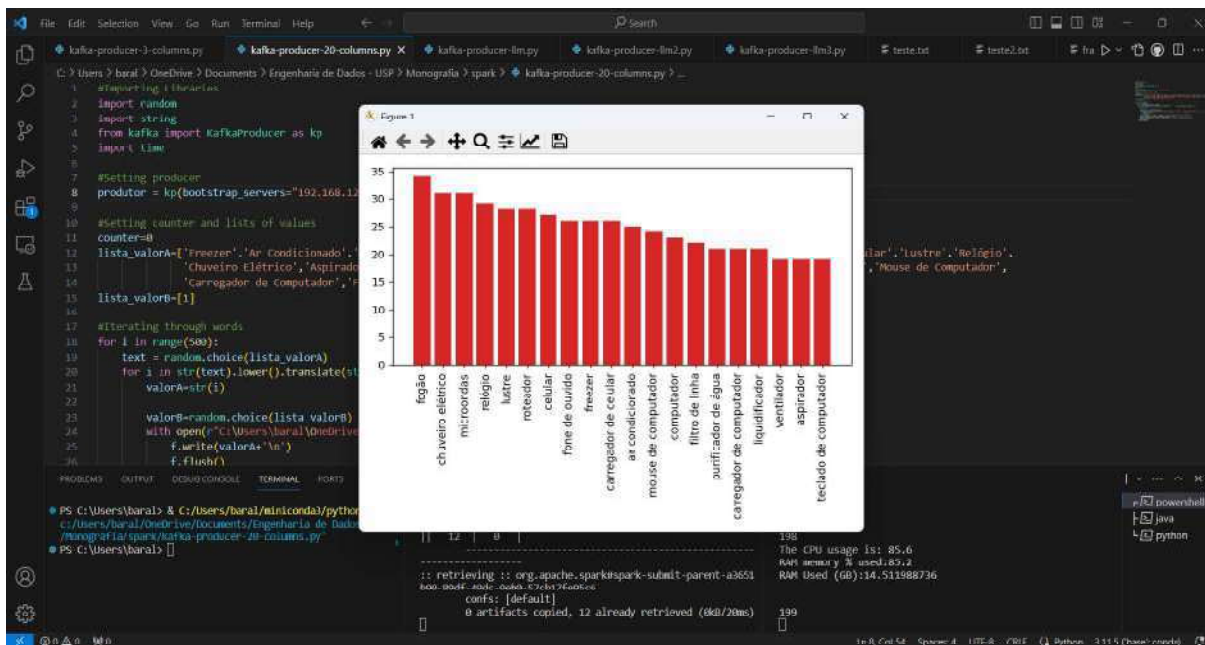
Tabela B-31 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
58,333	50,000	86,675	0,558

Fonte: criada pelo autor

B.2.2.1.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura B-59 - Visualização Final



Fonte: criada pelo autor

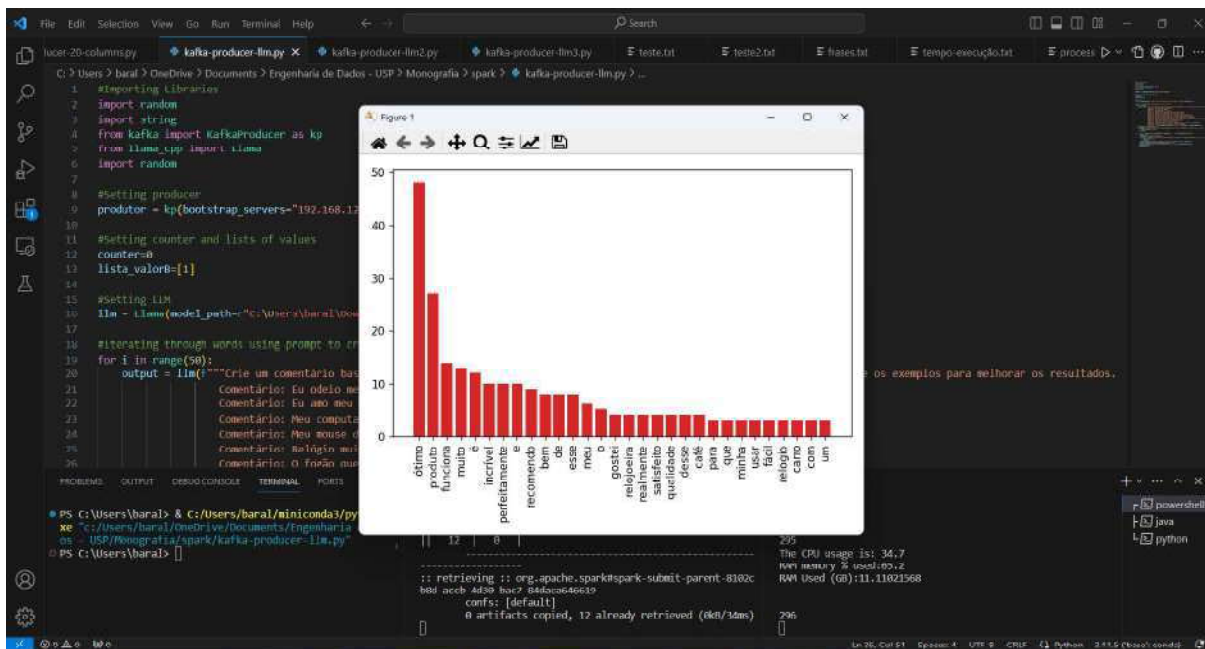
Tabela B-32 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
56,688	49,556	85,372	0,986

Fonte: criada pelo autor

B.2.2.1.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-60 - Visualização Final



Fonte: criada pelo autor

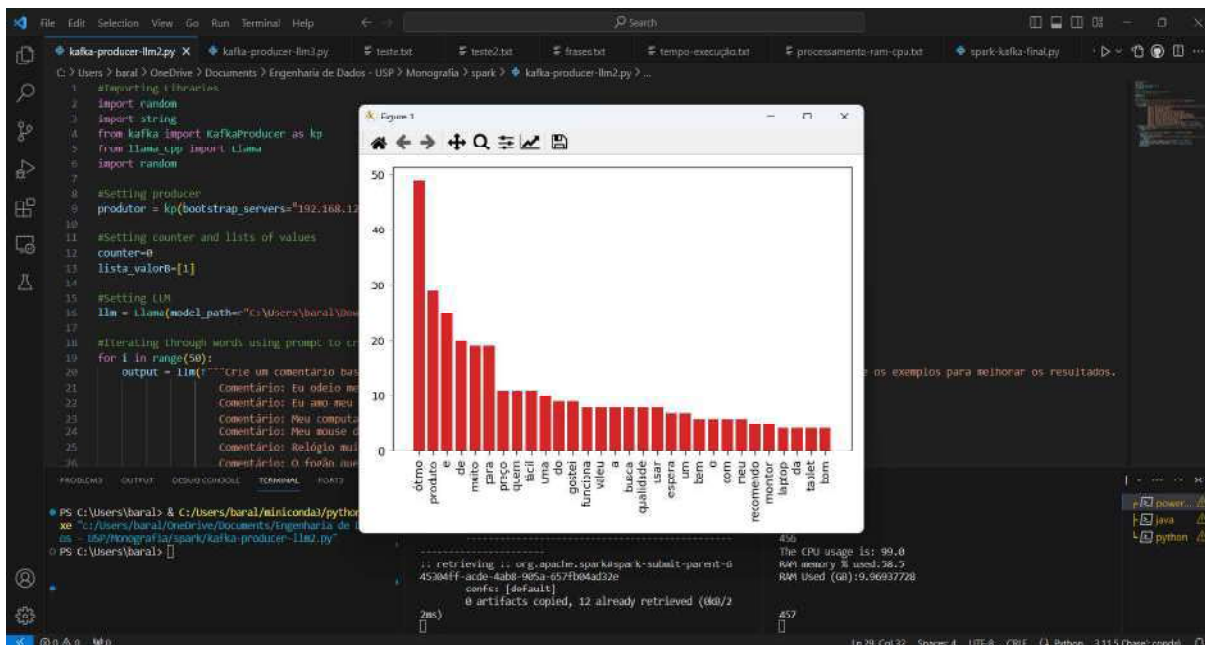
Tabela B-33 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
62,626	48,461	88,813	6,986

Fonte: criada pelo autor

B.2.2.1.4 THEBLOKE/LLAMA-2-13B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-61 - Visualização Final



Fonte: criada pelo autor

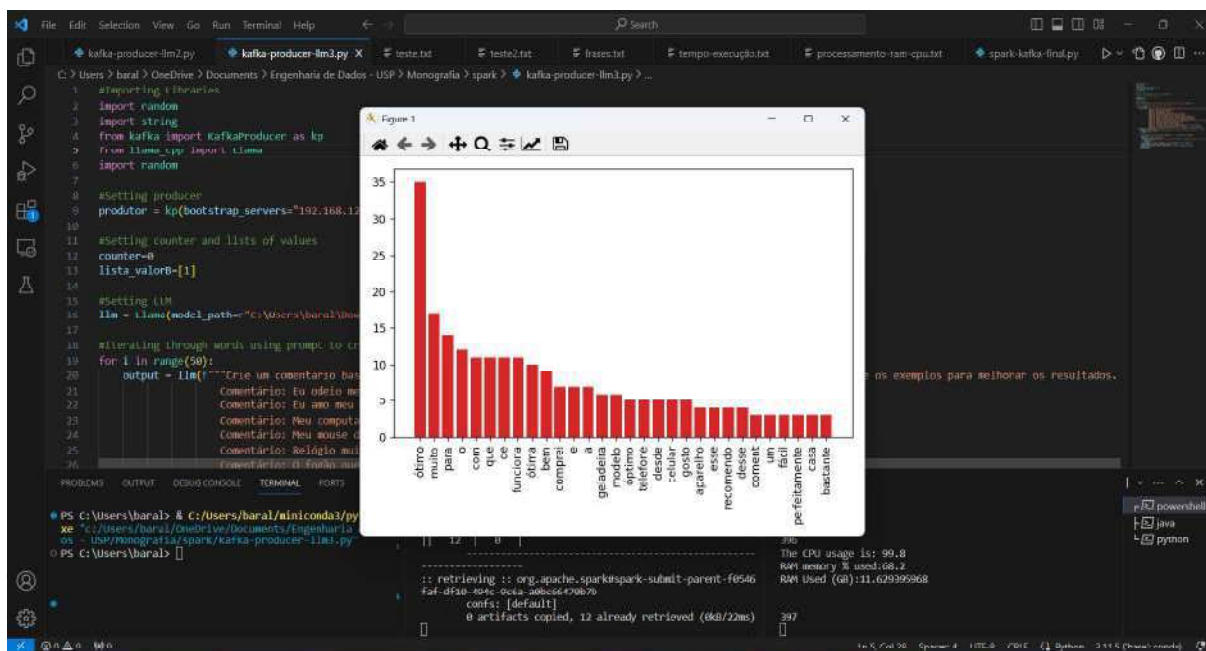
Tabela B-34 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
71,197	45,032	96,734	5,886

Fonte: criada pelo autor

B.2.2.1.5 THEBLOKE/ZEPHYR-7B-BETA-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-62 - Visualização Final



Fonte: criada pelo autor

Tabela B-35 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
70,184	45,741	91,344	6,254

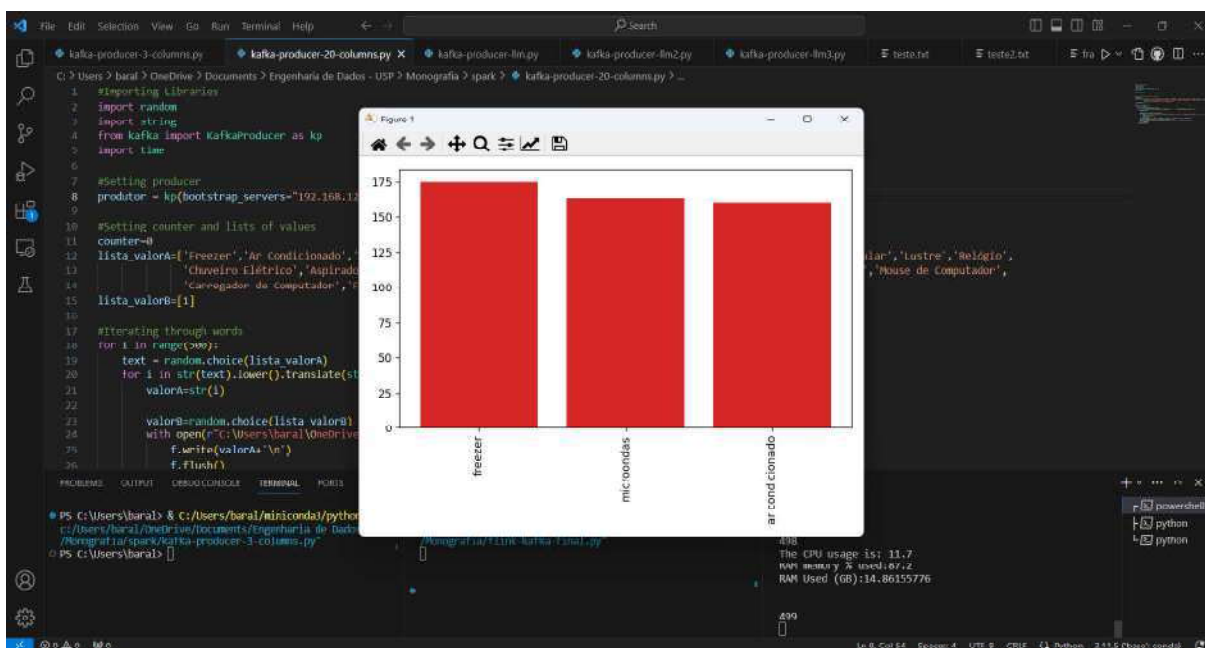
Fonte: criada pelo autor

B.2.2.2 PYFLINK

Cinco abordagens utilizando PyFlink.

B.2.2.2.1 TRÊS COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura B-63 - Visualização Final



Fonte: criada pelo autor

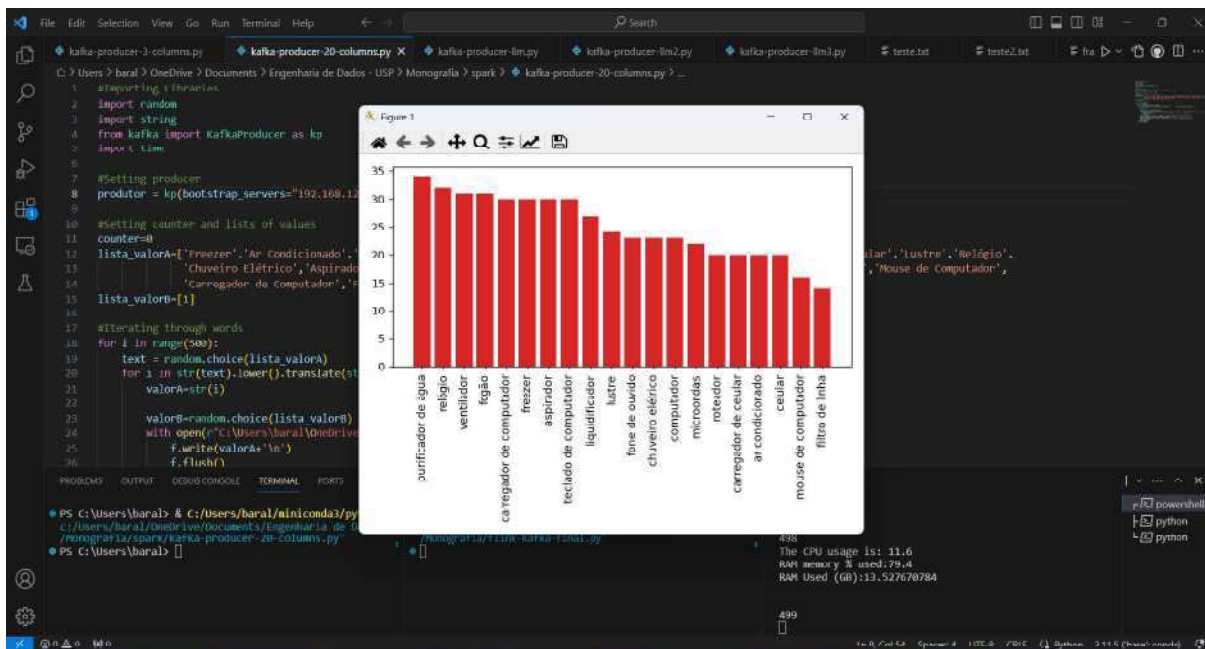
Tabela B-36 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
3,976	16,591	85,179	0,570

Fonte: criada pelo autor

B.2.2.2 VINTE COLUNAS/PRODUTOS (500 EXEMPLOS)

Figura B-64 - Visualização Final



Fonte: criada pelo autor

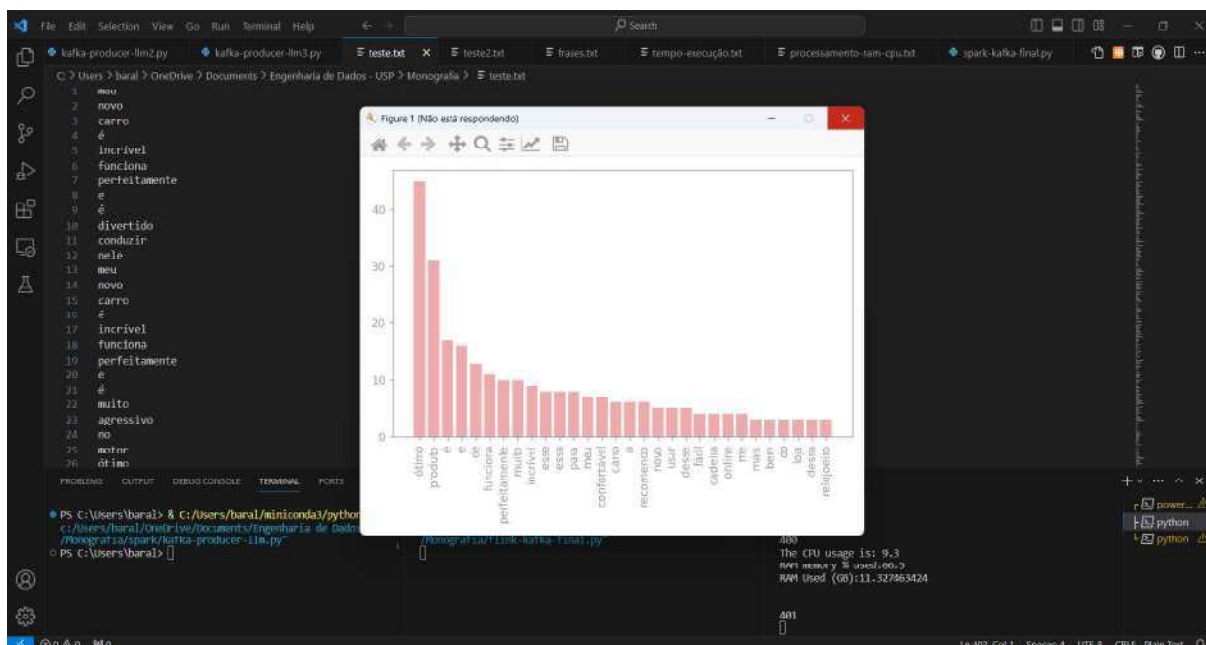
Tabela B-37 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
4,279	17,778	80,057	0,675

Fonte: criada pelo autor

B.2.2.2.3 THEBLOKE/LLAMA-2-7B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-65 - Visualização Final



Fonte: criada pelo autor

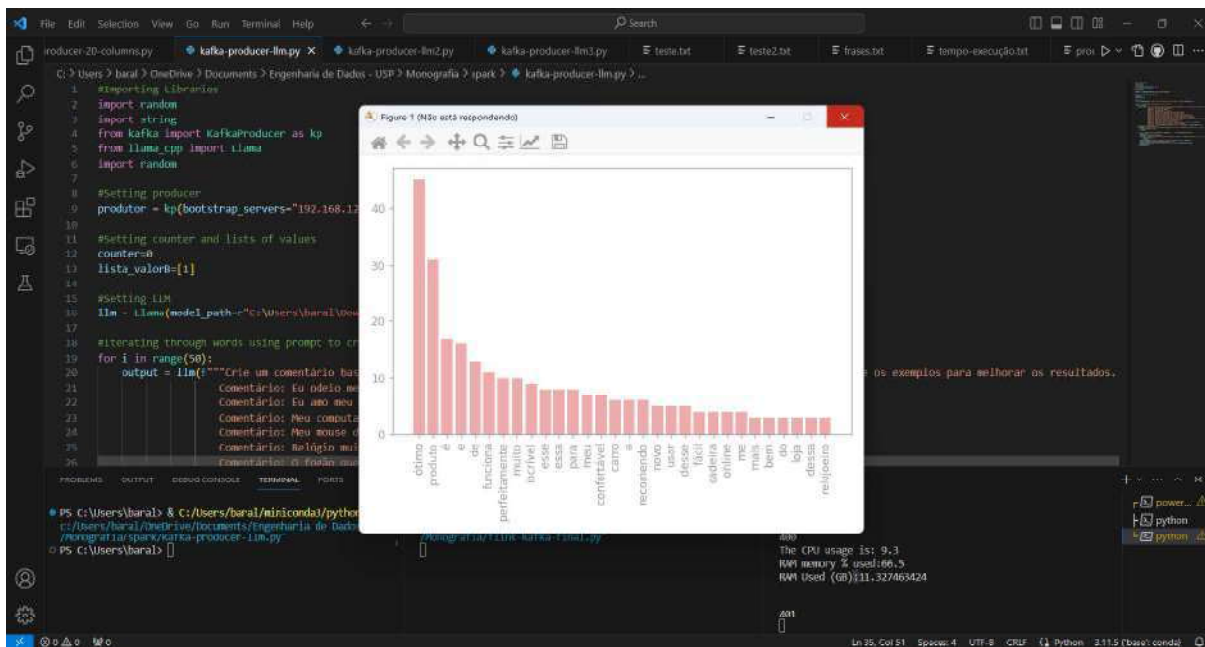
Tabela B-38 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
16,073	32,768	97,730	3,454

Fonte: criada pelo autor

B.2.2.2.4 THEBLOKE/LLAMA-2-13B-CHAT-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-66 - Visualização Final



Fonte: criada pelo autor

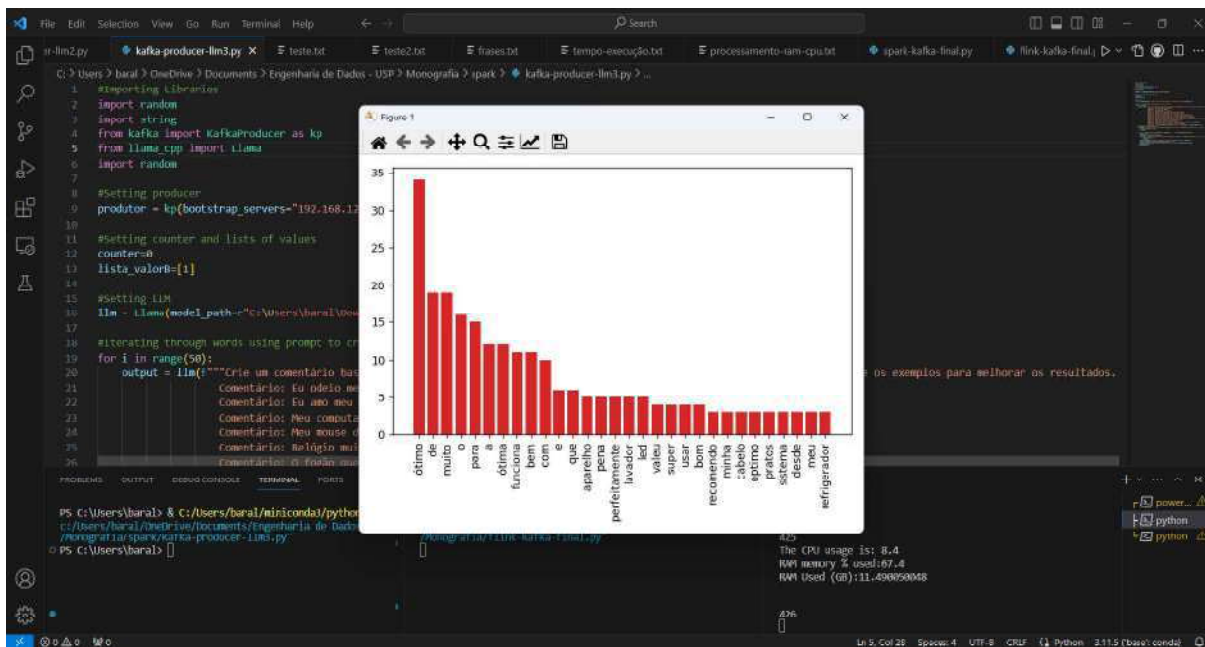
Tabela B-39 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
20,223	35,643	97,452	1,316

Fonte: criada pelo autor

B.2.2.2.5 THEBLOKE/ZEPHYR-7B-BETA-GGUF (50 GERAÇÕES DE TEXTO)

Figura B-67 - Visualização Final



Fonte: criada pelo autor

Tabela B-40 - Média e Desvio Padrão de Porcentagem de CPU e RAM

CPU %		RAM %	
MÉDIA	DESV. PADRÃO	MÉDIA	DESV. PADRÃO
10,059	23,752	96,036	4,590

Fonte: criada pelo autor