Abordagem de desenvolvimento de uma *pipeline* de migração de dados utilizando inteligência artificial

José Vítor Donassolo Correa dos Santos
Universidade Tecnológica Federal do Paraná
Dois Vizinhos, Brasil
josevitordonassolo@gmail.com

Evandro Miguel Kuszera
Universidade Tecnológica Federal do Paraná
Dois Vizinhos, Brasil
evandrokuszera@utfpr.edu.br

***Abstract*—Large Language Models (LLMs) and artificial intelligence are increasingly serving as tools to enhance productivity. Given the significant growth of data availability and the need to explore it, data migration became a pressing issue. This work presents an approach for generating code for data migration and transformation between relational databases, using LLMs and open-source tools that integrate the application with artificial intelligence models. As a result, the approach proved to be viable for generating data transformation pipelines, while also highlighting certain challenges with LLMs, which do not eliminate the need for a specialist in the migration process.**

***Keywords*—data engineering; artificial intelligence; natural language processing.**

***Resumo*—Os grandes modelos de linguagem (LLM) e a inteligência artificial estão cada vez mais servindo como ferramenta para auxiliar os usuários em diferentes funções e aumentar a produtividade. Tendo em vista também o grande crescimento da disponibilidade de dados e da necessidade de explorá-los, a manipulação de dados utilizando de técnicas de engenharia de dados é um assunto que se faz latente. Portanto este trabalho tem como finalidade unir estes dois aspectos para avaliar, através de diferentes critérios, a capacidade das ferramentas de código aberto que integram a aplicação com modelos de inteligência artificial em gerar códigos para migração e transformação de dados. A expectativa é que esta abordagem contribua para mitigar desafios latentes na área, como a escassez de profissionais qualificados no mercado de trabalho e a complexidade intrínseca a essas tarefas, que, com o emprego dessas técnicas, podem ser simplificadas.**

***Palavras-chave*—engenharia de dados; inteligência artificial; processamento de linguagem natural.**

I. Introdução

Com o aumento da disponibilidade de dados e a possibilidade de explorar o valor gerado por eles, profissões relacionadas a essa área estão em alta demanda no mercado de trabalho [1]. Um desses profissionais é o Engenheiro de Dados, que realiza a criação de *pipelines* de dados, permitindo que cientistas de dados trabalhem com dados limpos e organizados para análises, desenvolvimento de modelos de *machine learning* e *business intelligence* [2]. Desenvolver *pipelines* de transformação de dados pode ser uma tarefa desafiadora, ainda mais para profissionais que não têm conhecimentos técnicos aprofundados [3].

O problema que o trabalho aborda é a alta complexidade do desenvolvimento destas *pipelines*. E com o deficit de profissionais especializados conforme abordado no relatório da BRASSCOM de 2021 [4], torna-se essencial explorar alternativas viáveis que possam ser adotadas para enfrentar esse desafio e facilitar estes processos de desenvolvimento.

Nesse sentido, este trabalho apresenta uma abordagem para o desenvolvimento de uma *pipeline* de migração de dados utilizando ferramentas de inteligência artificial generativas através de *inputs* simples em linguagem natural, que possam ser solicitados por um desenvolvedor não especializado. Com base no *prompt* informado pelo usuário, a abordagem que utiliza-se de uma ferramenta de código aberto, que tem ampla contribuição da comunidade, e de ferramentas de inteligência artificial para a criação de *pipelines* de migração de dados em linguagem *Python* e códigos de SQL. O *prompt* do usuário é enriquecido com uso de técnicas de *prompt engineering* e dados extraídos dos metadados do banco de origem. Para avaliar a viabilidade da abordagem serão realizadas comparações entre as *pipelines* geradas pela IA e por um especialista.

O artigo está organizando da seguinte forma: na seção II são apresentados aspectos conceituais e na seção III os trabalhos relacionados. Na seção IV a abordagem proposta é apresentada. Na seção V são apresentados os experimentos e na seção VI as conclusões.

II. Aspectos Conceituais

A Inteligência Artificial (IA) visa desenvolver sistemas que imitam o raciocínio humano em tarefas de tomada de decisão e resolução de problemas [5]. Entre as suas categorias, destacam-se os *Large Language Models (LLM)*, projetados para compreender e gerar texto em linguagem natural [6]. Exemplos de LLMs incluem o GPT-4[1], *Google Gemini*[2] e o *Llama*, da

[1]https://openai.com/pt-BR/
[2]https://gemini.google.com/

Meta[3]. Os LLMs podem ser utilizados na migração de dados ao combiná-los com a utilização da *LangChain*[4], um *framework* em *Python* que facilita o desenvolvimento de projetos baseados em LLMs. O *LangChain* conecta-se a bancos de dados e faz requisições em linguagem natural aos modelos LLM, permitindo uma interação personalizada com dados estruturados.

Dentro deste contexto, se faz necessário também a utilização de técnicas de engenharia de *prompt* para obter melhores resultados dos retornos das ferramentas de inteligência artificial, fazendo perguntas melhores. O *prompt* é um conjunto de instruções que definem o contexto da conversa, quais informações são importantes e qual deve ser a forma e conteúdo da saída. A engenharia de *prompt* é o processo de programar as entradas para obter melhores resultados de saída, estabelecendo diretrizes que melhoram a qualidade das respostas geradas [7].

A migração de dados, necessária para integrar e analisar dados de diferentes fontes, requer tratamentos e transformações complexas [3]. O processo ETL (*Extract, Transform, Load*), fundamental para sistemas de *Business Intelligence*, é uma das etapas essenciais de um *pipeline* de dados mais abrangente [8]. O uso de LLMs pode ser um facilitador na construção de *pipelines*, auxiliando nas tarefas de ETL e análises de dados.

III. Trabalhos Relacionados

Foi realizada uma revisão da literatura buscando trabalhos que tenham relação com a proposta deste artigo. Como é o caso de [9] que estuda uma certa tradução das entradas em linguagem natural, para transformá-las em consultas ao banco de dados, na qual possibilitaria o usuário comunicar-se diretamente com a base de dados dessa maneira. O autor cita os desafios que são justamente relacionados a esse entendimento da linguagem natural e a necessidade de evolução para possibilitar uma maior flexibilidade, já que o trabalho deste autor não utiliza processamento de linguagem natural com inteligência artificial.

Em estudos que utilizam grandes modelos de linguagem e inteligência artificial para a geração de códigos de SQL, foi analisado o estudo [10], que anali-sa a capacidade de geração de SQL a partir de entradas com linguagem natural, utilizando principalmente o *ChatGPT*. No qual é passado um *prompt* padrão para o *ChatGPT*, solicitando apenas um SQL. A métrica de avaliação no trabalho [10] possui três aspectos: sintaxe do SQL, acurácia dos dados trazidos e a acurácia da execução em termos de padrão de código e desempenho. E concluiu que o *ChatGPT* chegou a bons resultados e conseguiu cumprir com a maioria das tarefas que foram propostas para desenvolvimento com base em entradas em formato de texto para SQL.

[3]https://llama.meta.com/
[4]https://www.langchain.com/

Tal qual o anterior, a monografia de [11] descreve também a respeito da tradução do texto em linguagem natural para SQL, este estudo se baseou em um conjunto de dados específico para realizar as solicitações para as ferramentas de inteligência artificial. A avaliação se deu com base no próprio método de avaliação oficial do conjunto de dados selecionado pelo autor e conclui que os modelos *open-source* não foram foram capazes de superar o modelo proprietário *ChatGPT*, que performou de maneira positiva em 39,9% do total dos casos, considerando todos os níveis de dificuldades levantados pelo estudo.

Dentro do atual estado da arte, o trabalho analisado que mais se aproximou com o objetivo da abordagem proposta neste artigo foi o *"Transforming Data Flow: Generative AI in ETL Pipeline Automatization"* [12], o qual também aborda diversas técnicas para automatizar a criação de estruturas de ETL a partir de solicitações em linguagem natural, e conclui com a indicação de que tal abordagem foi capaz de simplificar e reduzir o tempo de desenvolvimento de *pipelines* ETL, porém ainda apresenta limitações no desenvolvimento e atualização continua desta base.

Neste presente trabalho, o estudo será conduzido de forma que avalia a viabilidade da implementação do uso de modelos de LLMs *Open-Source* e proprietários, combinando-os com a utilização de uma *framework Open-Source* que vem sendo amplamente utilizada para a construção de aplicações baseadas em LLMs. Esta ferramenta realiza a conexão com o banco e faz a leitura dos metadados, para que a entrada por parte do usuário para a ferramenta de IA seja apenas a descrição da solicitação, e não precise detalhar aspectos como nomes de tabelas, nomes das colunas, tipo do banco de dados e relacionamento entre as tabelas, já que a ferramenta realiza esta integração. Dessa forma reduzindo grande parte da demanda necessária para implementação.

IV. Abordagem de Migração de Dados Usando LLMs

A Figura 1 apresenta uma visão geral da abordagem que utiliza ferramentas de inteligência artificial para desenvolvimento de SQL e ETL. Foi utilizado a ferramenta *LangChain* em conjunto com a API fornecida pelos LLMs (*ChatGPT-4*, *Gemini* e Llama) para a criação do SQL e do código *Python* que realiza a migração dos dados.

No passo 1 o usuário envia um *prompt* em linguagem natural para gerar uma *pipeline* de migração de dados entre uma base de origem e destino. No passo 2, o *prompt* é enviado a ferramenta *LangChain*, que coleta as informações estruturais do banco de dados (passo 3) e os envia como parâmetro para as LLMs de IA (passo 4) juntamente com o *input* do usuário. No passo 5 as LLMs realizam o processamento para realizar a criação do SQL para extrair os dados da base de origem. No

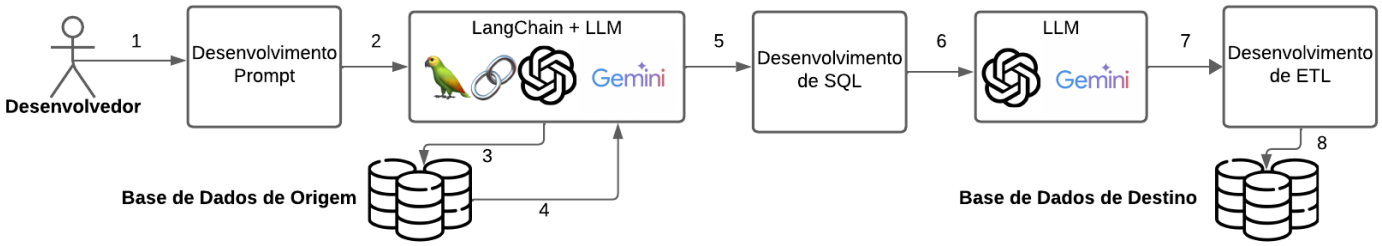

Fig. 1. Abordagem utilizando LLM

passo 6 esse SQL é fornecido como parâmetro para a LLM, que com base neste SQL irá gerar o código do ETL no passo 7, o qual permitirá que o usuário execute-o para popular estes dados na base de destino, realizando assim a migração de dados entre as bases.

A. *Base de Dados de Origem e Destino*

Para realizar a avaliação desenvolvida neste trabalho foram utilizadas duas bases de dados relacionais. Essas bases foram criadas usando o SGBD (Sistema de Gerenciamento de Banco de Dados) *PostgreSQL*. O *PostgreSQL* foi selecionado para o trabalho pois conforme estudos [13], é um dos melhores bancos de dados livres da atualidade, sendo largamente utilizado em diversas companhias. Na base de dados de origem foram criadas tabelas simulando um sistema de gestão de vendas, possuindo informações a respeito de cadastro de produtos, clientes, movimentações de compra e venda, e registro de financeiro, entre outras tabelas correlatas que complementam a informação. O modelo lógico do banco de dados é apresentado na Figura 2.

Após a criação da base de dados de origem foram inseridos em torno de um milhão de registros nas tabelas de movimentação e milhares de registros nas tabelas cadastrais, com informações fictícias para serem utilizadas na execução das migrações de dados.

A base de destino representa um *Data Warehouse*, seguindo o esquema apresentado na Figura 3. Os dados migrados pela abordagem serão persistidos com base nesse esquema.

B. *Prompt para criação do SQL*

Para definir o formato do *prompt* de entrada foram utilizadas técnicas de *Prompt Engineering*, então o *prompt* para criar o SQL estabeleceu-se da seguinte forma:

```
Retorne o texto do SQLQuery para uma
consulta que busque as colunas com os
dados:
INFORMACAO DESEJADA renomeando para NOME DA
COLUNA NA BASE DE DESTINO
```

O usuário deve inserir o nome da coluna que deseja do banco de origem e o nome da coluna na base de destino. Para cada coluna desejada deverá ser inserida uma nova linha neste formato. Para aplicar filtros deve ser inserido ao final do *prompt* a frase: “Buscar dados de determinado segmento”. Além do *prompt* e todas as informações de conexão com a base de origem, também é enviada uma mensagem padrão sugerida na documentação da *LangChain* para contextualizar os LLMs para gerar comandos SQL. O resultado dessa etapa é um comando SQL que recupera a informação desejada da base de origem.

Cada coluna solicitada no *prompt* deve corresponder a um campo da tabela na base de dados de destino, mantendo a quantidade de campos que nesta tabela existem, e as suas respectivas nomenclaturas. Portanto se o usuário inserir dados na tabela “DIM_NOTA_FISCAL”, apresentada na Figura 3, por exemplo, ele deve indicar qual campo corresponderá às colunas “nk_nota_fiscal”, “nr_nota_fiscal” e “ds_serie_nota_fiscal” respectivamente.

C. *Prompt para criação do ETL*

O *prompt* para a criação do ETL foi definido para que a solicitação do usuário seja a mais amigável possível, considerando usuário com baixo conhecimento técnico. O *prompt* padrão ficou definido da forma a seguir.

```
Com base nesse código: {sql}. Gere uma
pipeline de dados em python utilizando
pandas que insira estes dados em uma base
de destino com os seguintes dados de
acesso:{source_config}
Os dados da base de origem são os
seguintes: {target_config}
Tabela de destino: {nome_tabela} no schema
dw_ia.
Antes de inserir os dados na tabela,
deletar os  dados já existentes nela
utilizando um cursor do psycopg2 para
deleta-los.
```

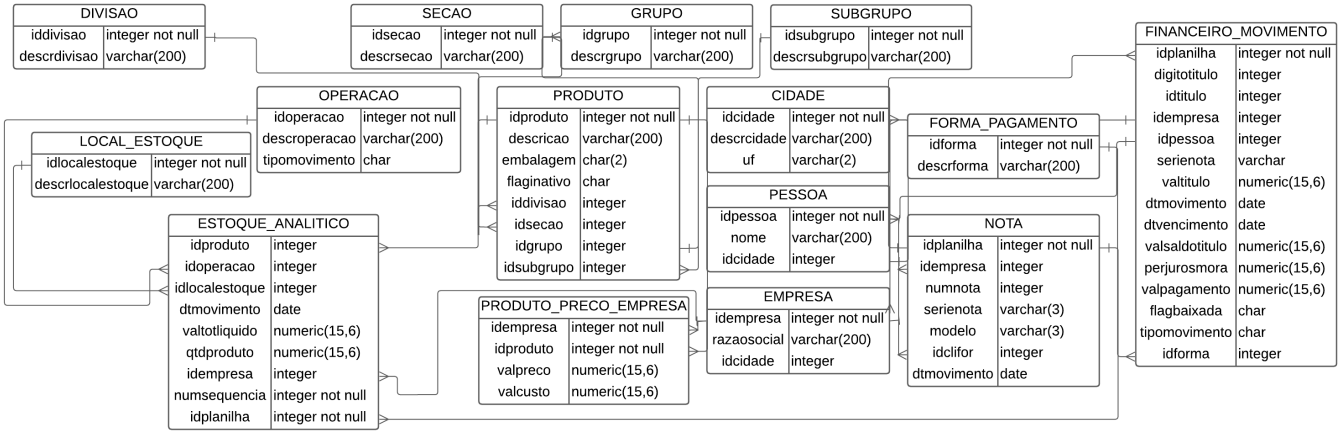

Fig. 2. Modelo Lógico Base de Origem

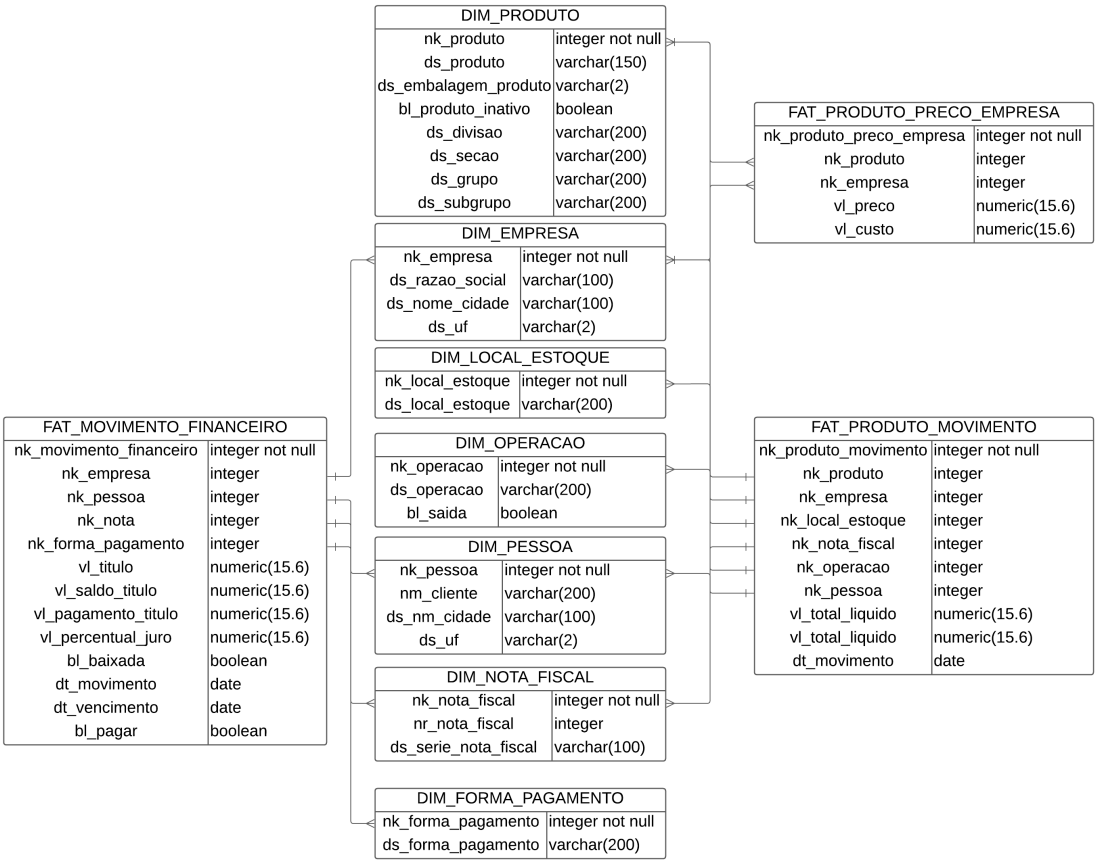

Fig. 3. Modelo Lógico Base de Destino

A variável identificada como "{sql}" irá receber o *output* retornado na etapa anterior com o código SQL gerado pela LLM. Nesta abordagem a única das variáveis que irá ser alterada pelo usuário em cada um dos casos é a variável nomeada como "{nome_tabela}", que representa a tabela na base destino. Com base neste código SQL, as variáveis de conexão da base de origem e destino e o nome da tabela em que os dados serão populados no destino, a LLM irá realizar o processamento para a criação do ETL, sendo este um *script* da linguagem *Python* para realizar a migração dos dados.

D. Configuração de acesso ao LLM

A abordagem tem integração com os modelos proprietários de LLMs *ChatGPT* e *Gemini* por meio de API KEY obtida no site das ferramentas. As versões utilizadas foram: *gpt-4-turbo* e o *gemini-pro*.

O modelo de código aberto *Llama* foi instalado na máquina local, em sistema operacional *MacOS* através do instalador baixado no site do *Ollama*.[5] Tendo sido instalado o modelo *llama3:8b*, na sua versão inferior, de 8 bilhões de parâmetros, em função das configurações de *hardware* da máquina em que foram desenvolvidos os experimentos. Tal máquina possui um *chip Apple M2* de 8 núcleos, com GPU integrada de 10 núcleos e memória *RAM* de 8 GB. [6]

Após a instalação do modelo na máquina que será utilizada para rodar as aplicações, não é necessário indicar nenhum *token* para utiliza-la, sendo necessário apenas fazer a declaração do modelo de *LLM* na aplicação.

V. Experimentos

Para avaliar a abordagem proposta neste artigo serão considerados cenários de migração de dados, em que as *pipelines* serão geradas com uso de LLMs. Foi abordado a utilização dos modelos proprietários *ChatGPT* e *Gemini*, os quais foram consumidos através das respectivas APIs fornecidas nos sites dos modelos, sendo que a do *ChatGPT* possui um custo variável por requisição, estipulado com base na quantidade de *tokens* e a do *Gemini* foi consumida de maneira gratuita. E também do modelo *Llama3.1*, desenvolvido pela Meta, e disponibilizada de forma gratuita para utilização, tal modelo foi instalado localmente na sua versão inicial, que possui 8 bilhões de parâmetros. Então estas estruturas de migração de dados geradas pelas ferramentas de IA foram comparadas com estruturas desenvolvidas manualmente por um analista com experiência e conhecimento das regras de negócio da base transacional.

Os critérios para realizar a avaliação comparativa entre a *pipeline* desenvolvida manualmente e a gerada pela IA são os seguintes: (i) Consistência dos dados migrados, através de comparações para verificar se as informações das tabelas de ambos os *schemas* são as mesmas; (ii) Tempo de execução da *pipeline*, que foi monitorado utilizando a biblioteca *timeit* da linguagem *Python*, que retorna o tempo médio de sete execuções e o desvio padrão entre elas; (iii) Custo do plano de acesso do SQL, que representa uma estimativa do tempo e recursos necessários para executar uma consulta e é uma métrica própria da base de dados que calcula com base na quantidade de dados das tabelas, relacionamentos e índices da consulta [14], é gerado no banco de dados *PostgreSQL* através da execução do comando *EXPLAIN* juntamente com o SQL; (iv) Quantidade de alertas retornados na ferramenta *Flake8*; (v) Por fim, o número de linhas do código, o que pode por vezes pode indicar a manutenibilidade de um código.

Os *prompts* usados nos experimentos e o código da *pipeline* gerado pela LLM estão disponíveis em repositório do *GitHub*[7]. Antes da execução de cada experimento todos os registros das tabelas da base de destino são removidos. O usuário deve fornecer o *prompt* de entrada, o nome da tabela na base de destino e os parâmetros de conexão com as bases de origem e destino. Três cenários diferentes foram definidos, nos quais a complexidade aumenta gradativamente. Foi indicado nos resultados de cada experimento quando se fez necessário ajustes no *prompt* para obter o retorno correto da LLM.

A. Cenário 1

Contexto: Buscar dados de apenas uma tabela, porém utilizando um filtro de período, solicitando que o SQL traga apenas dados referentes ao ano de 2023:

```
Retorne o texto do SQLQuery para uma
consulta que busque as colunas com os
dados:
Planilha da nota fiscal renomeando para
nk_nota_fiscal
Numero da nota fiscal renomeando para
nr_nota_fiscal
Serie da nota fiscal renomeando para
ds_serie_nota_fiscal
Busque apenas dados do período de 2023
```

Resultado: No primeiro cenário tanto o *ChatGPT* quanto o *Llama3.1* foram capazes de retornar o SQL correto. Sendo verificado apenas um desempenho menor no SQL retornado pela ferramenta *Open-source*, o que ocasionou em um maior tempo de execução. O *Gemini* tentou filtrar os dados utilizando a função *"strftime"* do *Python*, o que não é suportado pelo *PostgreSQL*, retornando assim um erro de execução. Na etapa

[5]https://ollama.com/
[6]https://support.apple.com/pt-br/111869
[7]https://github.com/Abordagemllm/abordagem-migracao-llm/tree/main

do *script* do ETL, novamente o *ChatGPT* e o *Llama3.1* foram capazes de retornar um código *Python* que corretamente deletou os registros da tabela antes de inserir e populou dados íntegros. Já o *Gemini* retornou um erro de sintaxe na execução do código ao tentar buscar os dados utilizando a função *to_sql* da biblioteca *pandas* utilizando um *cursor* do *psycopg2*, o que não é suportado. O código do *Gemini* apenas funcionou e rodou em tempo similar ao inserir a seguinte instrução: "*Para buscar os dados na tabela, utilize a função de conexão create_engine da biblioteca sqlalchemy*". A Tabela I apresenta os resultados para o Cenário 1, exibindo a LLM usada, o custo de execução do SQL (calculado pelo comando EXPLAIN ANALYSE do Postgres), o tempo médio de execução da consulta (8 execuções), número de alertas gerados por *Flake8* e número de linhas de código Python da *pipeline*. O item "*gemini-pro**" representa os resultados do *Gemini* com intervenções realizadas por um especialista para possibilitar a execução.

TABELA I
RESULTADOS DO CENÁRIO 1

LLM	Custo SQL	Tempo Médio Execução	Alertas *Flake8*	N. Linhas
gpt-4-turbo	19700	311 ms ± 39.5 ms	1	56
gemini-pro	-	-	1	55
gemini-pro*	19700	488 ms ± 80.2 ms	0	45
llama3.1	41000	745 ms ± 102.5 ms	5	68
Especialista	19700	486 ms ± 85.2 ms	0	16

B. Cenário 2

Contexto: O segundo cenário eleva o nível de complexidade do SQL, uma vez que utiliza relacionamentos entre tabelas e exige dos LLMs o entendimento das colunas que devem ser utilizadas. Novamente a etapa do código do ETL tem o mesmo requisito de deletar os dados da tabela de destino e inseri-los novamente. O dado buscado é o cadastro das empresas, junto das informações do nome da cidade e estado, que está armazenada em uma segunda tabela. Na solicitação o usuário indica que na base de destino os dados serão inseridos na tabela nomeada DIM_EMPRESA. *Prompt* utilizado:

```
Retorne o texto do SQLQuery para uma
consulta que busque as colunas com os
dados:
Codigo da empresa nomeando como
nk_empresa
Nome da empresa nomeando como
ds_razao_social
Nome da cidade renomeando como
ds_nome_cidade
Estado da cidade nomeando como ds_uf
```

Resultado: Este cenário teve o retorno correto do SQL em ambas as LLMs proprietárias utilizadas, porém o *Llama3.1* não foi capaz de realizar o relacionamento correto entre as tabelas que possuíam informações solicitadas e retornou erro de sintaxe por buscar colunas de tabelas não indicadas na *query*. Novamente o *Gemini* teve problemas na geração do Código do ETL, sendo necessário os mesmos ajustes para a geração do código correto. A Tabela II mostra os resultados obtidos. O item "*Llama3.1**" indica os resultados da *pipeline* fornecidos pelo *Llama3.1* com as correções aplicadas pelo especilista apenas na etapa do SQL, já que o código *Python* fornecido pelo modelo foi correto e funcional.

TABELA II
RESULTADOS DO CENÁRIO 2

LLM	Custo SQL	Tempo Médio Execução	Alertas *Flake8*	N. Linhas
gpt-4-turbo	217	85.3 ms ± 7.05 ms	1	60
gemini-pro	216	-	1	48
gemini-pro*	216	193 ms ± 13.9 ms	0	48
llama3.1	-	-	5	89
llama3.1*	216	201 ms ± 9.4 ms	5	90
Especialista	216	245 ms ± 156 ms	0	16

C. Cenário 3

Contexto: Neste cenário a consulta utiliza relacionamentos entre tabelas e requer dos LLMs o entendimento das colunas que devem ser utilizadas para relacionar elas. Para a etapa que busca o SQL é solicitado dados de movimentação de produtos, que requer o relacionamento de apenas duas tabelas, filtrando apenas dados movimentados nos últimos 2 meses. Na solicitação o usuário indica que na base de destino os dados serão inseridos na tabela nomeada FAT_PRODUTO_MOVIMENTO. *Prompt* utilizado:

```
Retorne o texto do SQLQuery para uma
consulta que busque as colunas com os
dados:
Planilha do movimento concatenada com o
numsequencia renomeando como
nk_produto_movimento
Código do produto renomeando como
nk_produto
Código da empresa renomeando como
nk_empresa
Planilha do lançamento renomeando como
nk_nota_fiscal
Código da pessoa nomeando como nk_pessoa
Valor total liquido nomeando como
vl_total_liquido
Quantidade nomeando como vl_qtd_produto
```

```
Data de movimento renomeando como
dt_movimento
Busque apenas dados dos ultimos 2 meses
```

Resultado: Neste cenário, nenhuma das LLMs obtiveram sucesso na geração do SQL e do ETL. O *ChatGPT* não identificou a tabela correta para retornar os dados da pessoa. Ele usou uma tabela que continha um campo nomeado como ”IDPESSOA”, que no contexto da consulta não fazia sentido. Esse problema ocorreu por que na base de dados de origem, a coluna que representa essa informação na tabela está nomeada como ”IDCLIFOR”, referindo-se ao código dos clientes e fornecedores, porém no contexto do *prompt* solicitado pelo usuário, ela representaria o dado do código da pessoa. A instrução SQL ficou correta depois de adicionar após a coluna a seguinte solicitação no *prompt*: ”(Buscando a informação de código da pessoa da tabela NOTAS, que na tabela está como IDCLIFOR)”. O *Gemini*, relacionou a tabela que possuía a coluna correta, porém fez tal relacionamento com as chaves incorretas, além de fazer a concatenação de colunas e buscar a função de data atual com a sintaxe incorreta. Na etapa de ETL, novamente o *Gemini* teve problemas na geração do código, sendo necessário o ajuste dos mesmos detalhes do cenário anterior para a geração do código correto. Por fim, o *Llama3.1*, retornou erro ao tentar utilizar como chave do campo identificador da pessoa a coluna ”NK_PESSOA”, a qual não existia em nenhuma das tabelas da base de dados, retornando assim um erro de execução já no SQL. Ao corrigir o SQL dessa ferramenta, notou-se também que o código do ETL gerado também não era funcional, uma vez que não inseriu corretamente os dados na base de destino.

A Tabela III mostra os resultados obtidos.

TABELA III
RESULTADOS DO CENÁRIO 3

LLM	Custo SQL	Tempo Médio Execução	Alertas *Flake8*	N. Linhas
gpt-4-turbo	-	-	7	70
gemini-pro	-	-	3	54
llama3.1	-	-	5	102
Especialista	95900	1.83 s ± 140 ms	0	63

D. Discussão dos Resultados

Com base nos experimentos desenvolvidos é possível observar que, tanto na geração de código SQL quanto a de ETL, estas ferramentas ficam atrás do desempenho de um especialista e ainda possuem débitos. No entanto, conseguem entregar grande valor na geração de código. Analisando mais especificamente por ferramenta, é possível ver uma maior proficiência do *ChatGPT* nos retornos de geração de SQL, que nas solicitações básicas tem desempenho similar ao do especialista e gera dados consistentes.

Porém também é notável uma eficiência relativa do modelo *Llama3.1*, o qual possuí número inferior de parâmetros em comparação com os demais modelos analisados [15], é *open-source* e conseguiu entregar um código correto em um cenário que um dos modelos proprietários não foi capaz. Portanto gerou grande valor mesmo quando não atendeu completamente a demanda.

O *Gemini* também foi capaz de gerar códigos corretos nas solicitações básicas, mas quando errou, ficou ainda mais distante de um retorno com sintaxe correta, trazendo por exemplo, colunas inexistentes na base de dados.

Na parte do ETL, para as migrações propostas nos dois primeiros cenários, somente o *ChatGPT* atendeu 100% as solicitações dos usuários, conseguindo migrar os dados de maneira precisa. Porém novamente é importante destacar a geração dos códigos do *Llama3.1*, o qual teve problemas nesta etapa apenas no último cenário, superando neste quesito, a ferramenta do *Gemini*. Apenas a manutenibilidade do código ficou abaixo ao compararmos com o do especialista, por conta da maior quantidade de linhas e repetição de funções parecidas nos diferentes códigos que poderiam ser modularizadas. Porém este aspecto ainda é uma limitação por conta da maneira que a ferramenta entrega o código. Nos cenários mais complexos ambas as ferramentas tiveram dificuldades.

Outro caso que leva a perda de desempenho das ferramentas é quando as nomenclaturas das colunas e tabelas não têm uma compreensão muito clara ao que elas se referem. Isso faz com que o código precise de adaptações para que possa ser um código funcional.

Ainda assim o ponto principal de contribuição da abordagem, é a facilidade da implementação das técnicas abordadas por utilizar-se de uma biblioteca que integra-se sem grandes dificuldades a bases de dados e aplicações, portanto mostrando-se viável a implementação destas técnicas em um processo de desenvolvimento, o qual pode ser otimizado.

VI. CONCLUSÃO

Neste projeto foi descrita uma abordagem de desenvolvimento de uma *pipeline* de dados através de entradas em linguagem natural para uma ferramenta que conecta-se com um banco de dados. Para isso foi realizado a revisão da literatura e estudo de todos os aspectos conceituais necessários para entendimento do assunto. Além disso, foi desenvolvido uma linha de abordagem para padronização das entradas de uma maneira que seja amigável a um usuário com baixo conhecimento técnico.

Os grandes modelos de linguagem que foram selecionados para a criação do código utilizado no processo de ETL foram o

Chat GPT-4, *Gemini-Pro* e *Llama3.1*, os quais foram testados em diversos cenários de geração de código para poder avalia-los e comparar a produtividade também entre os dois modelos. Visto que em uma realidade que os desenvolvedores já utilizam estas ferramentas, é também de extrema importância que isso seja avaliado para determinar uma preferência pelo uso de cada uma delas.

Neste trabalho os resultados indicam o valor agregado das ferramentas que foram apresentadas para aumentar a produtividade dos engenheiros de dados em tarefas de desenvolvimento de códigos de SQL e ETL mais simples que, muitas vezes podem consumir tempo, e com o uso das abordagens o trabalho é facilitado. Para usuários com menor conhecimento técnico de desenvolvimento, os resultados indicam principalmente a viabilidade de uma maneira de gerar instruções SQL que podem servir para a busca de dados necessários.

Além disso, ao analisar os resultados da abordagem, é possível inferir que há ganho de produtividade mesmo para profissionais especializados, que podem utilizar de tais técnicas para automatizar tarefas mais simples e realizar apenas uma revisão do código gerado antes de implementar, já que na maioria das vezes, conforme analisado no trabalho, os códigos gerados precisam de pouco ou nenhum ajuste, principalmente em cenários de baixa complexidade. Além disso, os profissionais especializados, em geral são capazes de fornecer melhores *prompts*, com mais detalhes e informações técnicas, que tendem aprimorar os retornos por parte das ferramentas. Porém tal conclusão pode ser ainda mais explorada em trabalhos futuros, analisando essa abordagem diretamente com profissionais que fizeram o uso desta.

O aumento da complexidade nos cenários indicou que ainda estas ferramentas não são capazes de substituir o trabalho de um especialista para o desenvolvimento, principalmente para os casos em que a base de dados não tem uma nomenclatura clara das tabelas e colunas e as regras de negócio são mais complexas. Nesses casos outras alternativas se fazem necessárias para viabilizar a abordagem.

Como trabalho futuro, pretende-se abordar aspectos como utilização de dicionários de dados como entrada para as ferramentas além de outras alternativas para contextualizar cada vez mais os *prompts*, além da utilização de outras ferramentas de inteligência artificial.

Referências

[1] Forbes. (2023) As 25 profissões em alta neste ano, segundo o linkedin. https://forbes.com.br/carreira/2023/01/as-25-profissoes-em-alta-neste-ano-segundo-o-linkedin/. Acessado em: 14 de novembro de 2023.

[2] J. ANDERSON. (2020) Data teams. https://www.datateams.io/wpcontent/uploads/2020/09/Anderson2020_Chapter_DataTeams.pdf. Acessado em: 12 de julho de 2024.

[3] M. Mendonça, "Metodologia de migração de dados em um contexto de migração de sistemas legados," Master's thesis, Universidade Federal de Pernambuco, 2009.

[4] Brasscom, "Demanda de talentos em tic e estratégia tcem." https://brasscom.org.br/pdfs/demanda-de-talentos-em-tic-e-estrategia-tcem, 2021, acessado em: 5 de setembro de 2023.

[5] . N. P. Russell, S., "Artificial intelligence: A modern approach," 1995.

[6] J. Okerlund, E. Klasky, A. Middha, S. Kim, H. Rosenfeld, M. Kleinman, and S. Parthasarathy, "Large language models, why they matter, and what we should do about them," University of Michigan, Tech. Rep., 2022, acesso em 14 de dezembro de 2022. [Online]. Available: https://stpp.fordschool.umich.edu/sites/stpp/files/2022-04/UM%20TAP%20Large%20Language%20Models%20Full%20Report%202022.pdf

[7] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," 2023. [Online]. Available: https://arxiv.org/abs/2302.11382

[8] M. Ross and R. Kimball, *The data warehouse toolkit: the definitive guide to dimensional modeling*. Wiley, 2013.

[9] R. A. Pazos R, J. J. González B, M. A. Aguirre L, J. A. Martínez F, and H. J. Fraire H, "Natural language interfaces to databases: an analysis of the state of the art," *Recent Advances on Hybrid Intelligent Systems*, pp. 463–480, 2013.

[10] A. Liu, X. Hu, L. Wen, and P. S. Yu, "A comprehensive evaluation of chatgpts zero-shot text-to-sql capability," 2023.

[11] G. d. M. Evangelista, "Uso de llm open source na tradução de linguagem natural para sql," B.S. thesis, 2023.

[12] C. V. der Putten, "Transforming data flow: Generative ai in etl pipeline automatization," Master Degree Thesis, Politecnico di Torino, 2024.

[13] C. C. Pimentel, "Bancos de dados relacionais: uma análise comparativa entre ferramentas sgbd livre e proprietária," *Tecnologia em Gestão da Tecnologia da Informação-Unisul Virtual*, 2019.

[14] Oracle. Understanding explain plan. https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i25909. Acessado em: 8 de junho de 2024.

[15] A. Souza, "Comparando capacidades de llms (large language models)," https://medium.com/blog-do-zouza/comparando-llms-large-language-models-945c9268c52f, 2023, acessado em: 30 de agosto de 2024.