

Cenários sob Demanda: uma Aplicação de LLM na Parametrização de um Otimizador Logístico na Minerva Foods

Jovânio José Galvão Júnior^{1,2}, Jorge Menezes dos Santos¹, Lucas Sirimarco Moreira Guedes¹, Rafael Ghussn Cano¹, Marina Chiara Hallak¹, Maria Tassiane Barros de Lima¹, Caio Perez Casagrande¹ e Rodrigo Valentim¹

¹ Minerva Foods, Barretos-SP, Brasil

{jovanio.galvao, jorge.santos, lucas.moreira, rafael.cano, marina.hallak, maria.lima, caio.casagrande, rodrigo.valentim}@minervafoods.com

² Instituto de Computação, Universidade Estadual de Campinas (UNICAMP), Campinas-SP, Brasil

RESUMO

Neste artigo é proposto um *framework* que integra modelos de linguagem de grande porte (LLM) com otimização logística, visando melhorar o planejamento de exportações da Minerva Foods. A ferramenta possibilita a geração de cenários via chat, permitindo que os usuários interajam e solicitem novos cenários em linguagem natural. O *framework* implementa uma arquitetura modular que simplifica o processamento de linguagem natural e melhora a rastreabilidade de falhas. Nesta fase de prova de conceito, foram avaliados o desempenho dos modelos gpt-3.5-turbo e gpt-4, que demonstraram desempenho robusto em tarefas simples e também para análises comparativas de cenários em contextos complexos derivados de casos reais. Em um conjunto de 45 perguntas, o *framework* obteve um erro médio de cerca de 3.5%. Com uma interface que traduz solicitações verbais em parâmetros quantificáveis, o sistema visa tornar o uso do otimizador mais acessível e eficaz, promovendo decisões baseadas em dados.

PALAVRAS CHAVE. LLM. Otimização logística. Análise de Cenários. Engenharia de prompt.

Tópicos: PO e IA; PO na Indústria

ABSTRACT

This paper presents a framework that integrates Large Language Models (LLMs) and logistics optimization, aiming to improve the export planning process at Minerva Foods. The proposed tool enables scenario generation via chat, allowing users to interact and request new scenarios in natural language. The framework implements a modular architecture that simplifies natural language processing and enhances error tracing and handling. In this phase, we developed a proof-of-concept prototype and evaluated the performance of both gpt-3.5-turbo and gpt-4 models, which demonstrated robust performance in simple tasks as well as in complex scenario analyses derived from real-world cases. In a set of 45 questions, the framework achieved an average error of approximately 3.5%. With an interface that translates user requests into quantifiable parameters, we aim at making optimizers more accessible and effective, promoting data-driven decision making.

KEYWORDS. LLM. Logistic Optimization. Scenario Analysis. Prompt Engineering.

Paper topics: OR and AI; Operations Research in Industry

1. Introdução

O planejamento de operações logísticas é um campo de ampla aplicação de técnicas de Pesquisa Operacional (PO) Azab et al. [2023]; Rejeb et al. [2023]. A otimização das opções de rotas de exportação da unidade de produção até o porto do país de destino, considerando os fretes terrestres e marítimos, é fundamental para a operação eficiente de uma grande empresa exportadora. Neste contexto, as equipes de PO e logística da Minerva Foods, uma das maiores exportadoras de carne bovina da América do Walzburiech [2024], desenvolveram uma ferramenta de otimização da logística de exportação.

Esta ferramenta de planejamento dá suporte tanto para planos semanais quanto mensais. O horizonte é flexível de acordo com as datas programadas de expedição das ordens e de *deadline* (a data limite para embarque de carga) dos navios nos terminais. Desta forma, há o uso de rotina, com foco na programação semanal e nas condições correntes do sistema logístico e suas restrições. Este foi o principal contexto de uso do sistema desde sua validação no início de 2024.

Entretanto, com a sua consolidação na rotina, a equipe de planejamento logístico da Minerva Foods identificou oportunidades de geração de cenários sob demanda, para análises estratégicas e de riscos. Diversos parâmetros podem ser alterados para avaliar os impactos e a sensibilidade das alocações das rotas. As alterações possíveis incluem desde mudanças nos valores de fretes e tarifas de diferentes serviços até oscilações nos tempos de deslocamentos terrestres (p. ex. impactos da malha logística por eventos climáticos), nas datas programadas de expedição (p. ex. impactos da produção) ou nas datas dos navios (p. ex. congestionamentos portuários).

A literatura recente sobre cadeias de exportação multi-modais evidencia ganhos substanciais quando modelos de programação inteira consideram simultaneamente custos e *leadtime* (tempo total de transporte), especialmente em sistemas integrados mar-ferrovia Abu-Aisha et al. [2024]; de Faria et al. [2024]. Além disso, abordagens de roteamento robusto em redes intermodais sob cenários de ruptura reforçam a necessidade de ferramentas que permitam gerar e avaliar rapidamente múltiplos cenários “e se” Uddin e Huynh [2016].

Paralelamente, pesquisas que integram Large Language Models (LLM) a métodos de Pesquisa Operacional — como o *OptiGuide* Li et al. [2023] ou o sistema de preferência iterativa para agendamento Lawless et al. [2023] — apontam para um novo paradigma de interação entre usuários de negócio e algoritmos de otimização, no qual consultas em linguagem natural são traduzidas em ajustes paramétricos.

A partir deste novo paradigma, diversas investigações mostram que a simples modificação do *prompt* pode potencializar ou comprometer de forma drástica a ação dos LLMs sobre dados estruturados. Estudos em domínios sensíveis, como psiquiatria, provam que trocar uma única palavra pode inverter por completo a recomendação gerada Grabb [2023]. No extremo oposto, um prefácio tão curto quanto “Let’s think step by step” já duplicou a acurácia em tarefas aritméticas Kojima et al. [2022], e técnicas de *auto-consistência* acrescentaram até 18 p. p. adicionais Wang et al. [2022]. Métricas formais, como o *PromptSensiScore* no *framework* PROSA Zhuo et al. [2024] e o *dataset* PROMPTSET Razavi et al. [2025], revelam que paráfrases triviais ainda conseguem alterar decisivamente as respostas de modelos de última geração. A biblioteca PROMPTBENCH institucionaliza essa avaliação em larga escala Zhu et al. [2024], enquanto abordagens de *prompting is programming* advogam instruções declarativas para reduzir ambiguidades Beurer-Kellner et al. [2023]. Em paralelo, técnicas de compressão de *prompt* permitem instruções mais enxutas sem perda de qualidade Li et al. [2024]. Do ponto de vista da segurança, ataques de *prompt injection* exigem filtros antes de executar tarefas Liu et al. [2024], e levantamentos de engenharia de *prompt* fornecem diretrizes para raciocínio robusto Qiao et al. [2022]. Em conjunto, esses resultados convergem para a conclusão de que a qualidade do *prompt* é determinante para a confiabilidade dos cenários logísticos gerados.

Neste contexto, foi desenvolvida uma prova de conceito para integrar em um ambiente de chat, modelos LLM com o otimizador logístico, para criar e executar cenários alternativos de planejamento a partir de requisições em linguagem natural do usuário (conhecidas como questões do tipo “e se”). A partir de um cenário base já otimizado e salvo no banco de dados da aplicação, o usuário poderá enviar requisições em formato de texto para alterar diversos parâmetros do cenário e re-executar o otimizador.

O *framework* proposto é similar ao apresentado por Li et al. [2023], que para responder a consultas hipotéticas sobre otimização de cadeias de suprimentos, também usa o LLM para modificar o otimizador e, em seguida, executar novamente. As alterações do LLM são geradas diretamente no código do otimizador. Um agente programador baseado em LLM também é aplicado no *framework* proposto por Lawless et al. [2023], para lidar com preferências do usuário na automação de agendamentos de reuniões por programação por restrições. O código, em python, para adicionar, retirar ou alterar uma restrição é gerado pelo agente a partir da requisição do usuário.

Na abordagem desenvolvida não há geração de código direto no otimizador. Essa opção por manipular apenas o *payload* JSON, em vez de gerar código, dialoga com resultados de avaliações de qualidade de código produzido por LLMs, que sugerem combinar geração automática e camadas de verificação para minimizar erros Coignion et al. [2024]; Barke et al. [2024].

Cada requisição de alteração de parâmetros de um cenário é formalizada como uma função python para alteração do *payload* do otimizador. Para transformar pedidos de alteração por linguagem natural em funções python, aplicam-se modelos LLMs para traduzir mensagens de chat em operações. Esta resposta do interpretador LLM é repassada em formato estruturado e não diretamente em linguagem natural, com o objetivo de evitar redundâncias e estruturas sintáticas complexas da linguagem natural, e também reduzir o uso de tokens [Bacchin, 2024; Chen et al., 2024].

Já a manipulação de fato do arquivo JSON é realizada posteriormente por uma funcionalidade do *backend* do *framework*, que também gerencia as requisições à API do otimizador e controla a qualidade dos retornos dos modelos de LLM. Desta forma, o *framework* proposto é genérico, podendo ser adaptado para outros otimizadores, adaptando as instruções e conferências para o *payload* de cada caso.

O restante do documento se organiza da seguinte forma. Na Seção 2, o otimizador logístico é descrito resumidamente. A Seção 3 apresenta o *framework* desenvolvido, com detalhes dos *prompts* e organização do fluxo de uso. Na Seção 4, os resultados dos experimentos computacionais são apresentados e discutidos. Por fim, a Seção 5 discute as conclusões do trabalho.

2. Otimizador Logístico

Dados um conjunto de ordens de exportação divididas em dois grupos (A e B), as rotas terrestres das unidades de produção até os portos, as rotas de cada navio e a capacidade disponível por navio e armador, o otimizador tem dois objetivos: (i) alocar o grupo A minimizando o *leadtime*; (ii) alocar o grupo B minimizando a soma dos custos de fretes terrestres e marítimos. Devido à diferença de objetivos e primazia do grupo A sobre o grupo B, há dois problemas sequenciais de otimização de rotas completas de exportação, compostas pelo frete terrestre, terminal portuário, navio e armador. Além destes, também abordamos um subproblema associado ao tratamento e relaxação de restrições de cota mínima de uso de alguns fornecedores. Deve-se respeitar a cota, porém, caso seja inviável atingi-la, deve-se aproximar o máximo dela.

Uma visão de alto nível do procedimento completo é apresentada no Algoritmo 1. Inicialmente, é realizada a alocação otimizada do grupo A utilizando um algoritmo de fluxo em redes. Em seguida, relaxamos as restrições de cota mínima considerando a alocação do grupo A e as opções para o grupo B, através de um modelo de PLI. O objetivo é que toda rodada do otimizador forneça uma solução aceitável, ainda que algumas dessas restrições não sejam totalmente satisfeitas. De-

pois, é realizada a otimização das ordens do grupo B, também por meio de um modelo de PLI. Por fim, as soluções parciais são concatenadas e a solução final retornada.

Algorithm 1: Visão geral do otimizador logístico.

- 1 Seja *data* um objeto contendo todos os dados de entrada;
 - 2 $\text{sol_grupo_A} \leftarrow \text{otimizacao_rede_A}(\text{data})$;
 - 3 $\text{ajuste_cota_min} \leftarrow \text{relax_cota_min}(\text{data}, \text{sol_grupo_A})$;
 - 4 $\text{sol_grupo_B} \leftarrow \text{otimizacao_IP_B}(\text{data}, \text{ajuste_cota_min}, \text{sol_grupo_A})$;
 - 5 $\text{solucao_final} \leftarrow \text{sol_grupo_A} \cup \text{sol_grupo_B}$;
 - 6 **return** (*solucao_final*);
-

A etapa de minimização do *leadtime* para o grupo A é baseada no algoritmo de fluxo máximo e busca binária, atualizando iterativamente a rede de fluxo com as opções com *leadtime* menor ou igual ao parâmetro de busca *k*. O objetivo é encontrar o menor valor de *k* tal que o fluxo máximo seja equivalente ao tamanho do conjunto de ordens do grupo A.

Para otimizar a alocação do grupo B, o modelo decide qual rota completa é selecionada para cada ordem. A função objetivo é o somatório dos custos das rotas selecionadas (frete terrestre e marítimo e taxas portuárias) e o custo de não envio das ordens (penalização). As restrições garantem que: (i) no máximo uma rota completa seja selecionada para cada ordem, (ii) a quantidade de ordens alocadas em cada dupla (navio, armador) seja menor ou igual à capacidade disponível (descontada a alocação do grupo A) e (iii) o respeito às cotas mínimas de envio aplicadas a certos tipos de rotas (já considerando o valor ajustado se necessário).

O otimizador foi desenvolvido como um serviço (API) do sistema de planejamento logístico de exportação da Minerva Foods. O usuário, através da interface conectada ao banco de dados, parametriza os dados necessários para a execução do otimizador. Esses dados são enviados para a API no formato de um arquivo JSON padronizado. O resultado com as alocações das ordens, as rotas completas e os custos é retornado também em JSON padronizado. Ambos os arquivos, de dados de entrada e da solução, são salvos no banco do otimizador.

3. Framework proposto para geração de cenários via chat

A dinâmica de geração de cenários para o otimizador logístico foi mapeada e foram identificados os tipos de cenários comumente demandados pela área de negócio. Desta forma, foi proposto um *framework* para habilitar a interação direta do usuário, requisitando novos cenários e execuções através de comandos por linguagem natural em ambiente de chat. A visão geral desse *framework* é apresentada na Figura 1.

O *framework* implementa uma arquitetura de decomposição de responsabilidades utilizando múltiplos estágios de *prompts* e validações especializadas. Esta abordagem modular não apenas simplifica a complexidade inerente ao processamento de linguagem natural, mas também facilita a manutenção e o ajuste fino em cada etapa.

A decomposição melhora a rastreabilidade do processo de interpretação, possibilitando a identificação precisa de pontos de falha. Desta forma, cada retorno do modelo de LLM é seguido de uma etapa de validação e controle do fluxo do processo (etapas representadas por losangos na Figura 1), para avaliar a consistência e plausibilidade dos retornos.

Em caso de retornos fora do esperado, mensagens padrão são enviadas no chat para orientar o usuário a reescrever a requisição. Na Figura 1 essas mensagens estão junto aos pontos destacados em vermelho. Há também um ponto de contato intermediário para validação das alterações requisitadas, destacado pelos pontos azuis na Figura 1. Optou-se pela comunicação em formato de tabelas, apresentando o parâmetro a ser alterado com o valor atual e o valor modificado. O parâmetro e valor modificados são extraídos pela terceira etapa de execução do modelo LLM.

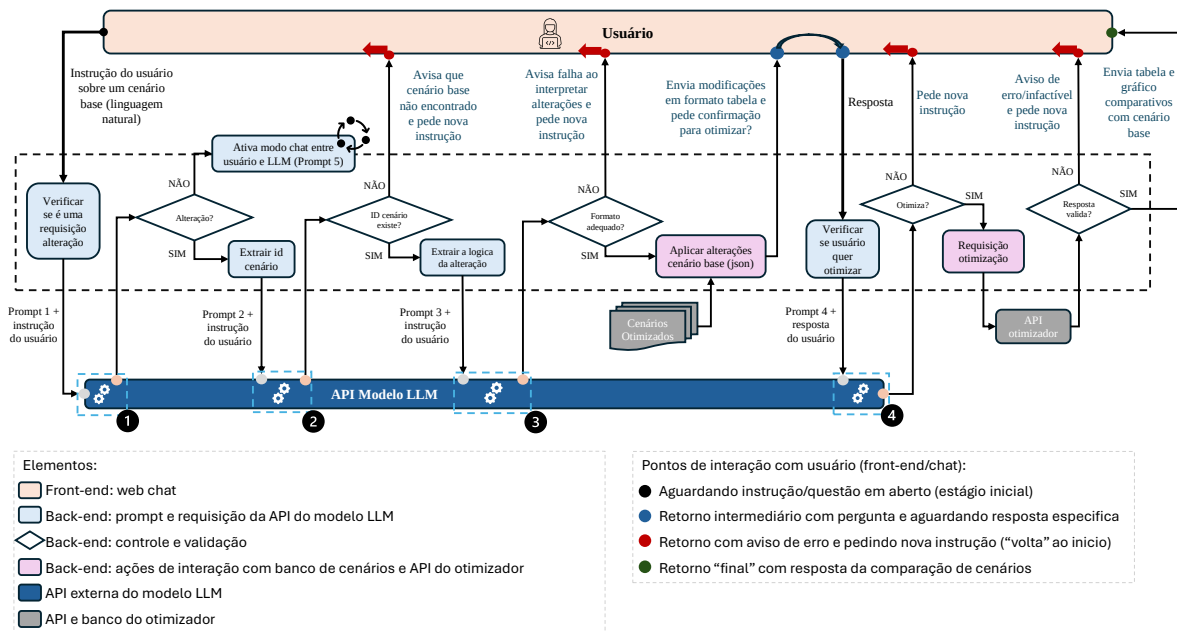


Figura 1: Framework de chat iterativo para alterações de cenários de otimização logística baseado em LLM.

Essa estratégia, composta de ponto intermediário e camadas de validação no *backend*, foi projetada para minimizar a propagação de erros e alucinações do modelo LLM ao usuário final. Outro fator para reduzir ambiguidades e potencializar a taxa de sucesso nas respostas geradas pelo modelo LLM foi o desenvolvimento de *prompts* especializados para cada etapa.

Cada *prompt* é projetado para executar uma função específica e bem definida no *pipeline* de processamento: (i) classificação inicial da requisição, (ii) extração de identificadores do banco de cenários, (iii) análise das alterações solicitadas e transformação no formato para funções Python, (iv) validação de respostas e (v) orientação para iterações com usuários do otimizador logístico. O objetivo é implementar *prompts* que assegurem uma interpretação clara e objetiva dos comandos em linguagem natural submetidos pelos usuários.

3.1. Prompts auxiliares

No primeiro caso, o objetivo é um retorno lógico (verdadeiro ou falso) do modelo LLM, sendo verdadeiro para mensagens do usuário que se referem a um cenário. Neste *prompt*, por exemplo, é ressaltado que o usuário poderia escrever a palavra cenário de forma errada, passando diversas grafias, mas também informa que outras palavras poderiam ser usadas nesse contexto com o mesmo significado de cenários, tais como: execução, rodada, simulação.

Já na segunda etapa, o *prompt* orienta que apenas um retorno numérico seja extraído da mensagem como identificador do cenário (ou vazio caso não encontre). Esta chave será usada para recuperar o JSON do cenário base no banco do otimizador. Neste momento, há uma oportunidade para exemplificar os formatos esperados dessas chaves no sistema de otimização logística.

O *prompt* da etapa 4 pede para identificar se a resposta do usuário é positiva (ou negativa) para seguir com a otimização do novo cenário. Por fim, o *prompt* 5 estabelece um agente que deve se comportar como um assistente inteligente do sistema de otimização logística da Minerva Foods. Sua função é responder perguntas gerais sobre o propósito, funcionamento e uso do sistema, bem como orientar o usuário a limitar suas consultas a cenários específicos, já que questões gerais sobre

o processo de otimização estão fora da sua área de atuação.

3.2. Processando alterações

O *prompt* 3 é responsável por identificar as alterações e transformá-las em comandos em python para posterior aplicação. A primeira parte estabelece a visão geral da tarefa: “Você receberá uma pergunta do usuário para alterar ou consultar dados em um *payload* que será manipulado localmente. Não envie nem modifique o *payload* completo; retorne APENAS as instruções mínimas em formato JSON para que o código local aplique as alterações.”. Então, define-se o formato padrão do retorno:

```
{
  "key": <chave do dado, ex: "frete_maritimo", "moedas" ou
  "taxa_juros", etc.>,
  "instruction_to_execute": <codigo Python para realizar a
  modificacao ou consulta no lado local>,
}
```

A premissa base é a geração de instruções para a manipulação de um *dataframe* pandas, informada no *prompt* da seguinte forma: “Assumindo que extrairemos os dados associados à chave e os usaremos pra criar um DataFrame do pandas chamado *df*, esse campo deve conter a instrução em python para realizar a modificação. Ex:”

```
df.loc[df['ID_Armador']==1001, 'valor_frete_maritimo'] *= 1.1 ou
df.loc[df['Cod_Servico'] == 'NAME', 'Sc_enable'] = False
```

Em alguns casos, o dado manipulado não é armazenado em um *dataframe*. Tais exceções são tratadas conforme ilustrado no exemplo: “Se a chave do dado for *taxa_juros* não teremos um DataFrame, mas apenas um dicionário com uma única chave. Então, o campo *instruction_to_execute* deve considerar um dicionário chamado *data*. Exemplo: *data[taxa_juros] = 14.25*.”

A formatação de campos específicos também é repassada, como no caso de datas. “Se for necessário alterar ou inserir uma data, use o formato “YYYY-MM-DDTHH:MM:SS”(exemplo: “2025-01-13T00:00:00”)”

Foram adicionadas instruções para orientar na validação, por exemplo: “Se a pergunta mencionar um ‘destino’ ou outro valor crítico, verifique os dados disponíveis para garantir que o valor informado esteja correto. Por exemplo, compare com os ‘Destinos das ordens’ e ‘Destinos gerais’. Caso o valor informado não exista nos dados do cenário, mas haja algum valor semelhante nos dados, use esse valor no lugar. Tome cuidado com acentos no nome das colunas dos DataFrames, observe os metadados disponíveis para isso, praticamente nenhuma coluna possui nomes com acentuação.”

Junto a estas instruções, metadados de cada cenário são enviados para consulta e validação. Esses metadados contêm não apenas um resumo do JSON a ser manipulado, mas também todos os valores únicos presentes no cenário base, organizados de forma mais simples e direta. Dessa forma, o modelo LLM pode capturar, mais eficientemente, informações como: moedas utilizadas, IDs das ordens, destinos envolvidos, modais de transporte, tamanhos de contêineres e outros pontos relevantes. Dessa forma, tem-se uma visão completa de tudo o que existe dentro do *payload*, sem precisar enviar o arquivo completo para o modelo LLM.

Por fim, algumas informações adicionais são repassadas, como conteúdo de chaves (ex: “A chave *cod_porto_destino* se refere ao código do porto de destino. Cada porto possui vários terminais.”) e um dicionário de chaves numéricas do banco de dados para armadores, portos e terminais com os respectivos nomes comerciais usados pelos usuários do sistema.

3.3. Exemplos

Alguns exemplos de requisições do usuário para novos cenários são apresentados em conjunto com o retorno do modelo LLM com a função python para alterar o *payload*, para facilitar a compreensão do funcionamento do *framework*.

3.3.1. Exemplo 1

“Reduza em 5% a cotação do dólar no cenário 1”

O primeiro *prompt* retornará verdadeiro devido o contexto da frase esta relacionado a um cenário. E em seguida, será extraído a chave 1 para recuperar o cenário no banco, através da segunda execução do modelo LLM.

Aplicando o terceiro *prompt*, o campo (key) moedas no *payload* é identificado e a moeda dólar americano (USD) é selecionada para fazer a operação de redução em 5% do valor (equivalente a multiplicar por 0.95) no *dataframe*:

```
{ 'key': 'moedas',
  'instruction_to_execute':
  ‘df.loc[df['moeda'] == 'USD', 'valor'] *= 0.95’}
```

Desta forma, o *backend* irá manipular de fato o *dataframe* (convertido a partir do JSON do cenário 1) com este comando repassado pelo modelo de LLM. Caso essa alteração não gere nenhum erro de execução, será enviado ao chat do usuário a Tabela 1 para a confirmação.

Moeda	Valor base	Novo valor
USD	5,700	5,415

Tabela 1: Tabela resumo das alterações do Exemplo 1.

A resposta do usuário será avaliada no *prompt* 4, e caso confirme a alteração, é feito a requisição na API do otimizador e enviado a resposta comparativa para o usuário ao final do processo.

3.3.2. Exemplo 2

“O que acontece com o cenário 3 se planta A não puder usar ferroviário?”

Aplicando o terceiro *prompt*, o campo (key) tempo frete terrestre no *payload* seria identificado para a alteração. Esta consistiria em retirar as opções definidas como multimodais para esta unidade do *dataframe*. Desta forma, não haveria fretes com ferrovia saindo da planta A para um porto. O resultado esperado, considerando a chave numérica da planta A igual a 1, seria:

```
{ 'key': 'tempo_frete_terrestre',
  'instruction_to_execute':
  ‘df.loc[(df['ID_Unidade'] == 1)
  & (df['is_multi_modal'] == 'Multimodal'),
  'is_multi_modal'] = None’}
```

Devido à limitação de páginas, não foi possível apresentar detalhadamente as perguntas neste artigo. Por esse motivo, disponibilizamos um arquivo público contendo a lista completa de perguntas utilizadas nos testes¹.

¹<https://drive.google.com/drive/folders/1jKYZGMaYj1pPfOzxmfz6f5fS-ZMhrEVR?usp=sharing>

4. Testes

Todos os experimentos foram realizados em um *notebook* Dell equipado com processador Intel Core i7-12700H, 40 GB de memória RAM (4800 MHz) e sistema operacional Ubuntu 22.04. As rotinas foram implementadas em Python 3.11 por meio da API do ChatGPT OpenAI [2023b], avaliando-se os modelos gpt-3.5-turbo OpenAI [2023c] e gpt-4 OpenAI [2023a].

4.1. Metodologia de criação das instâncias

Partiu-se de nove perguntas-base, distribuídas em dois grupos: **G1** (Questões 1-4), consideradas de menor complexidade, e **G2** (Questões 5-9), de maior complexidade. Para cada pergunta foram geradas cinco variações redacionais. Cada variação foi submetida a 31 execuções em cada modelo, perfazendo

$$9 \text{ perguntas} \times 5 \text{ variações} \times 31 \text{ execuções} \times 2 \text{ modelos} = 2\,790 \text{ testes.}$$

Como o tempo de resposta individual foi irrelevante, não se considerou o fator tempo na análise.

4.1.1. Critério de avaliação

As saídas do assistente foram classificadas em: *corretas*, quando a resposta satisfaz integralmente a solicitação; *incorretas*, quando falha totalmente; e *parcialmente corretas*, quando atendem apenas parte do requisito. Considere, por exemplo, dois *dataframes* — `df_ordens` (`id_ordem`, `destino_ordem`) e `df_destinos` (`destino`, `pais`). Se a tarefa exige remover de `df_ordens` as ordens cujo destino seja “Shanghai” e o assistente também remove a linha correspondente em `df_destinos`, o objetivo prático é alcançado, mas a operação adicional é desnecessária; classifica-se, portanto, como acerto parcial. Resultados parciais não são contabilizados nem como acertos, nem como erros.

4.1.2. Apresentação dos resultados

Os resultados consolidados dos testes encontram-se nas Tabelas 2 e 3. A Tabela 2 discrimina o desempenho por pergunta, grupo e modelo; a Tabela 3 resume os percentuais por grupo. Uma planilha detalhada — com todas as 2 790 instâncias (grupo, pergunta, variação, execução e modelo) — permanece disponível em planilha pública².

²<https://docs.google.com/spreadsheets/d/1fK1vimNPuXoo4hM5TxAQ43-jjNqEL7kRznQfNH1aFVw>

Tabela 2: Resultados por pergunta, grupo e modelo

Grupo	Pergunta	Erros %	Acertos %	Parciais %	Modelo
G1	P1	0.00	100.00	0.00	gpt-3.5-turbo
G1	P1	0.00	100.00	0.00	gpt-4
G1	P2	7.74	91.61	0.65	gpt-3.5-turbo
G1	P2	0.65	99.35	0.00	gpt-4
G1	P3	20.00	80.00	0.00	gpt-3.5-turbo
G1	P3	12.26	87.74	0.00	gpt-4
G1	P4	0.00	100.00	0.00	gpt-3.5-turbo
G1	P4	0.00	99.35	0.65	gpt-4
G2	P5	0.00	100.00	0.00	gpt-3.5-turbo
G2	P5	0.00	90.32	9.68	gpt-4
G2	P6	1.94	62.58	35.48	gpt-3.5-turbo
G2	P6	0.65	80.65	18.71	gpt-4
G2	P7	6.45	0.65	92.90	gpt-3.5-turbo
G2	P7	17.42	0.00	82.58	gpt-4
G2	P8	0.00	100.00	0.00	gpt-3.5-turbo
G2	P8	0.00	100.00	0.00	gpt-4
G2	P9	0.00	100.00	0.00	gpt-3.5-turbo
G2	P9	0.00	100.00	0.00	gpt-4

Tabela 3: Resultados consolidados por grupo

Grupo	Erros %	Acertos %	Parciais %	Modelo
G1	6.94	92.90	0.16	gpt-3.5-turbo
G1	3.23	96.61	0.16	gpt-4
G2	1.68	72.65	25.68	gpt-3.5-turbo
G2	3.61	74.19	22.19	gpt-4

4.1.3. Análise dos resultados

Os resultados obtidos no **G1** ratificam a elevada precisão nos itens de menor complexidade. O gpt-3.5-turbo alcançou 92.9 % de acertos, com 6.9 % de erros e apenas 0.2 % de resultados classificados como *parcialmente corretos*. O gpt-4 elevou a taxa de acerto para 96.6 %, reduzindo os erros a 3.2 % e mantendo a mesma fração residual de saídas parciais. Observa-se que as falhas concentram-se em P3 (20 % de erros no gpt-3.5-turbo e 12.3 % no gpt-4), ao passo que P1 e P4 continuam resolvidas com 100 % de sucesso em ambos os modelos, e P2 apresenta pequena discrepância entre as arquiteturas (91.6 % x 99.3 %).

Em **G2** verifica-se mudança substancial de perfil: a taxa de erros cai para 1.68 % no gpt-3.5-turbo e 3.61 % no gpt-4, porém os acertos parciais sobem para 25.7 % e 22.2 %, respectivamente, situando a acurácia em 72.6 % e 74.2 %. Isto indica que, diante de solicitações mais intrincadas, os modelos raramente falham por completo; com maior frequência, entregam a resposta correta acompanhada de operações supérfluas, enquadrando-se na categoria *parcialmente correta*. O padrão torna-se explícito em P6 e, sobretudo, em P7: nesta última questão, 92.9 % das saídas do gpt-3.5-turbo e 82.6 % das do gpt-4 obtiveram classificação parcial, com

virtual ausência de erros e de acertos plenos.

Em síntese, os resultados confirmam a robustez de ambos os modelos em tarefas simples e revelam, nas tarefas complexas, a substituição de erros por respostas funcionalmente corretas porém excessivas. O gpt-4 preserva vantagem modesta (≈ 1.5 p.p.) sobre o gpt-3.5-turbo, enquanto reduz levemente a incidência de saídas parciais. Ressalta-se que as respostas excessivas, classificadas como parciais, são úteis na geração dos cenários e posterior avaliação dos resultados comparativos, não comprometendo as análises do ponto de vista de negócio.

4.1.4. Sensibilidade dos modelos ao *prompt*

Os resultados apresentados já correspondem à versão refinada dos *prompts*. Convém, porém, registrar que, na fase inicial dos testes, a Questão P7 apresentou desempenho significativamente inferior: 100 % das respostas do gpt-3.5-turbo e 85.16 % das do gpt-4 estavam integralmente incorretas. Esse quadro só foi revertido após uma revisão sistemática do *prompt*, na qual se empregaram técnicas de compressão e reescrita orientadas à tarefa Beurer-Kellner et al. [2023]. Após a reformulação, a taxa de erro caiu para 6.45 % no gpt-3.5-turbo e 17.42 % no gpt-4; a maioria das saídas passou a ser classificada como parcialmente correta (82–92 %). Em termos absolutos, mais de 60 p.p. de respostas que antes falhavam completamente tornaram-se, no mínimo, funcionalmente úteis. Esse ganho confirma a literatura sobre engenharia de *prompts*: alterações mínimas de redação podem inverter diagnósticos Grabb [2023], ao passo que instruções passo a passo simples aumentam a acurácia em tarefas estruturadas Kojima et al. [2022]; Wang et al. [2022]. No domínio logístico, tal sensibilidade implica que a confiabilidade de análises “*what-if*” depende não apenas da qualidade do algoritmo de otimização, mas também do rigor com que o *prompt* delimita o espaço de busca. O desenho do *prompt*, portanto, deve ser tratado como um artefato de projeto — sujeito a controle de versão, experimentação A/B e validação contínua — e não como mero detalhe textual.

5. Conclusões

No *framework* proposto, a integração de LLMs com sistemas de pesquisa operacional permite uma interação direta de usuários leigos com otimizadores, através da criação de cenários sob demanda. Os modelos LLM, com sua capacidade avançada de processamento de linguagem natural, atuam como mediadores entre o usuário e os algoritmos de otimização, traduzindo solicitações verbais em parâmetros quantificáveis. Desta forma, essa interface torna o sistema mais acessível e eficaz, permitindo que decisões baseadas em dados sejam dirigidas não só por inputs técnicos, mas também descritas em termos naturais. Ao alavancar essa sinergia, o nosso *framework* potencializa a adaptabilidade do otimizador nos ambientes dinâmicos e complexos da logística da Minerva Foods.

Combinando *prompts* estruturados e validações, o *framework* busca minimizar os riscos associados a interpretações errôneas, promovendo uma interação mais confiável entre o usuário e o sistema. Nesta primeira prova de conceito, os modelos da OpenAI, gpt-3.5-turbo e gpt-4, foram utilizados para as tarefas de interpretação das requisições do usuário, e estes alcançaram resultados promissores nas perguntas mais diretas (Grupo 1) e também no Grupo 2, considerando acertos parciais, que são respostas úteis para o usuário final avaliar o resultado do cenário de negócio.

Os próximos passos serão levar a proposta para interação direta com usuários em fase de testes e avaliar melhorias de *prompt*, escalabilidade e consumo de tokens (considerando impactos nos custos da ferramenta). Também serão testados outros modelos, como opções open-source. A evolução deste projeto também contará com outros otimizadores da Minerva Foods, explorando a arquitetura de serviços. Neste processo, será necessário evoluir nas metodologias de testes com possível contribuição de *benchmarks* públicos para avaliação de performance.

Referências

- Abu-Aisha, T., Audy, J.-F., e Ouhimmou, M. (2024). Toward an efficient sea-rail intermodal transportation system: A systematic literature review. *Journal of Shipping and Trade*, 9(1):23.
- Azab, R., Mahmoud, R. S., Elbehery, R., e Gheith, M. (2023). A bi-objective mixed-integer linear programming model for a sustainable agro-food supply chain with product perishability and environmental considerations. *Logistics*, 7(3):46.
- Bacchin, F. (2024). The rise of llm-powered agents: Reinventing workflow automation with agentic ai. Master degree in computer science, University of Padova, Itália. URL <https://hdl.handle.net/20.500.12608/84815>.
- Barke, S., Anaya Gonzalez, E., Kasibatla, S. R., Berg-Kirkpatrick, T., e Polikarpova, N. (2024). Hysynth: Context-free llm approximation for guiding program synthesis. *Advances in Neural Information Processing Systems*, 37:15612–15645.
- Beurer-Kellner, L., Fischer, M., e Vechev, M. (2023). Prompting is programming: A query language for large language models. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1946–1969.
- Chen, W., Yuan, C., Yuan, J., Su, Y., Qian, C., Yang, C., Xie, R., Liu, Z., e Sun, M. (2024). Beyond natural language: Llm leveraging alternative formats for enhanced reasoning and communication. URL <https://arxiv.org/abs/2402.18439>.
- Coignon, T., Quinton, C., e Rouvoy, R. (2024). A performance study of llm-generated code on leetcode. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, EASE '24, p. 79–89, New York, NY, USA. Association for Computing Machinery. ISBN 9798400717017. URL <https://doi.org/10.1145/3661167.3661221>.
- de Faria, C. H. F., Almeida, J. F. F., e Pinto, L. R. (2024). Simulation–optimisation approach for sustainable planning of intermodal logistics in the brazilian grain export industry. *Decision Analytics Journal*, 10:100388.
- Grabb, D. (2023). The impact of prompt engineering in large language model performance: a psychiatric example. *Journal of Medical Artificial Intelligence*, 6(0). URL <https://jmaia.meagroups.org/article/view/8190>.
- Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., e Iwasawa, Y. (2022). Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.
- Lawless, C., Schoeffer, J., Le, L., Rowan, K., Sen, S., St.Hill, C., Suh, J., e Sarrafzadeh, B. (2023). ”i want it that way”: Enabling interactive decision support using large language models and constraint programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM.
- Li, B., Mellou, K., Zhang, B., Pathuri, J., e Menache, I. (2023). Large language models for supply chain optimization. URL <https://arxiv.org/abs/2307.03875>.
- Li, Z., Liu, Y., Su, Y., e Collier, N. (2024). Prompt compression for large language models: A survey. *arXiv preprint arXiv:2410.12388*.

- Liu, Y., Jia, Y., Geng, R., Jia, J., e Gong, N. Z. (2024). Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium (USENIX Security 24)*, p. 1831–1847.
- OpenAI (2023a). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*. URL <https://arxiv.org/abs/2303.08774>.
- OpenAI (2023b). Openai api. <https://platform.openai.com/docs>.
- OpenAI (2023c). Openai api. <https://platform.openai.com/docs/api-reference/introduction>.
- Qiao, S., Ou, Y., Zhang, N., Chen, X., Yao, Y., Deng, S., Tan, C., Huang, F., e Chen, H. (2022). Reasoning with language model prompting: A survey. *arXiv preprint arXiv:2212.09597*.
- Razavi, A., Soltangheis, M., Arabzadeh, N., Salamat, S., Zihayat, M., e Bagheri, E. (2025). Benchmarking prompt sensitivity in large language models. In *European Conference on Information Retrieval*, p. 303–313. Springer.
- Rejeb, L., Chaabani, A., Safi, H., e Ben said, L. (2023). Multimodal freight transport optimization based on economic and ecological constraint. In *Advances in Computational Logistics and Supply Chain Analytics*, p. 99–127. Springer.
- Uddin, M. M. e Huynh, N. (2016). Routing model for multicommodity freight in an intermodal network under disruptions. *Transportation Research Record*, 2548(1):71–80.
- Walzburiech, D. (2024). Quais são os maiores frigoríficos de carne bovina do brasil? *Globo Rural*. URL <https://globorural.globo.com/pecuaria/noticia/2024/11/quais-sao-os-maiores-frigorificos-de-carne-bovina-do-brasil.ghhtml>. Acesso em: 14 ago. 2025.
- Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., e Zhou, D. (2022). Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.
- Zhu, K., Zhao, Q., Chen, H., Wang, J., e Xie, X. (2024). Promptbench: A unified library for evaluation of large language models. *Journal of Machine Learning Research*, 25(254):1–22.
- Zhuo, J., Zhang, S., Fang, X., Duan, H., Lin, D., e Chen, K. (2024). Prosa: Assessing and understanding the prompt sensitivity of llms. *arXiv preprint arXiv:2410.12405*.