UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO

EDEILSON COSTA DE AZEVEDO FILHO
THIAGO RIBEIRO DA SILVA

ANÁLISE COMPARATIVA DE DESEMPENHO ENTRE LLM'S NO DESENVOLVIMENTO DE UM SISTEMA FUZZY DE AVALIAÇÃO DE RISCO EM PROJETOS DE SOFTWARE

INTELIGÊNCIA ARTIFICIAL

Maceió, Alagoas
2024

1. CONTEXTUALIZAÇÃO

O presente relatório documenta o processo de desenvolvimento e análise de dois sistemas baseados em regras fuzzy para a avaliação de risco em projetos de *software*. Cada sistema foi desenvolvido utilizando a abordagem de *pair programming* com o suporte de diferentes modelos de linguagem natural (LLMs). O objetivo principal deste trabalho é comparar o desempenho das LLMs no contexto de desenvolvimento colaborativo, bem como avaliar a eficácia dos sistemas criados para o diagnóstico de riscos em projetos de software.

Os sistemas foram projetados com base em 12 variáveis fuzzy e um conjunto de regras pré-definido, gerado por uma LLM de qualidade superior (GPT-4), garantindo uma base sólida para a construção das avaliações de risco. Essa abordagem permitiu explorar não apenas a implementação técnica, mas também o impacto da colaboração com as LLMs no desempenho e na qualidade do software produzido.

Em seguida, serão apresentadas as metodologias de desenvolvimento, detalhando as etapas seguidas em comum para ambos os sistemas. Posteriormente, será realizada uma análise individual dos resultados obtidos com a primeira LLM (GPT-4) e a segunda LLM (Gemini), culminando em um comparativo que destaca as diferenças e similaridades entre os desempenhos das duas tecnologias

2. METODOLOGIA

O desenvolvimento deste projeto foi estruturado em três principais etapas, possibilitando a organização e facil entendimento dos processos durante a construção dos *softwares*. Cada etapa será detalhada nos subtópicos deste referencial metodológico.

2.1. DEFINIÇÃO DE VARIÁVEIS E REGRAS

A primeira etapa do desenvolvimento consistiu na definição das variáveis fuzzy que seriam utilizadas para a avaliação do risco nos projetos de software. Essas variáveis foram escolhidas com base em fatores típicos que influenciam o risco de projetos, como complexidade, prazo, recursos, e experiência da equipe. Foram selecionadas 12 variáveis fuzzy de entrada, que seriam responsáveis por representar as diversas facetas do risco do projeto e 1 variável de saída, representando um valor, em porcentagem, para o risco do projeto.

NOME	VALOR	INTERVALO
Experiência da equipe	Anos	1 - 10
Nível de entrosamento	Escala qualitativa	0 - 10
Rotatividade da equipe	% de substituições	0 - 100
Definição de requisitos	Escala qualitativa	0 - 10
Mudanças nos requisitos	% de alterações significativas	0 - 100
Complexidade Tecnica	Escala qualitativa	0 - 10
Tamanho do projeto	Pontos de função	0 - 100
Prazo	Meses	1 - 36
Comunicação	Escala qualitativa	0 - 10
Maturidade das ferramentas	Escala qualitativa	0 - 10
Impacto do ambiente externo	Escala qualitativa	0 - 10
Orçamento disponível	USD	5.000 - 100.000
Risco	% de risco no projeto	0-100

Tabela 1. Variáveis de entrada e saída para o problema

Em seguida, foram gerados conceitos *fuzzy* para cada variável, necessários para a determinação das funções de pertinência que possibilitam a existência dos conjuntos *fuzzy*.

1. Experiência da Equipe (anos) [1, 10]

- **Baixa** [1, 4]
- **Média** [3, 7]
- **Alta** [6, 10]

2. Nível de Entrosamento da Equipe (escala subjetiva) [0, 10]

- **Baixo** [0, 3]
- **Médio** [2, 7]
- **Alto** [6, 10]

3. Rotatividade da Equipe (% de substituições) [0, 100]

- **Baixa** [0, 20]
- **Moderada** [15, 60]
- **Alta** [50, 100]

4. Definição de Requisitos (escala subjetiva de clareza) [0, 10]

- **Mal Definida** [0, 4]
- **Moderada** [3, 7]
- **Bem Definida** [6, 10]

5. Mudanças nos Requisitos (% de alterações significativas) [0, 100]

- **Baixas** [0, 20]
- **Moderadas** [15, 60]
- **Altas** [50, 100]

6. Complexidade Técnica (escala subjetiva de complexidade) [0, 10]

- **Baixa** [0, 3]
- **Média** [2, 7]
- **Alta** [6, 10]

7. Tamanho do Projeto (pontos de função) [0, 100]

- **Pequeno** [0, 30]
- **Médio** [20, 70]
- **Grande** [60, 100]

8. Prazo (meses) [1, 36]

- **Curto** [1, 6]
- **Moderado** [5, 18]
- **Longo** [16, 36]

9. Comunicação (escala subjetiva de qualidade de comunicação do grupo) [0, 10]

- **Ruim** [0, 3]
- **Boa** [2, 7]
- **Ótima** [6, 10]

10. Maturidade das Ferramentas (escala subjetiva) [0, 10]

- **Imatura** [0, 3]
- **Moderada** [2, 7]
- **Alta** [6, 10]

11. Impacto do Ambiente Externo (escala subjetiva) [0, 10]

- **Baixo** [0, 3]
- **Moderado** [2, 7]
- **Alto** [6, 10]

12. Orçamento Disponível (USD) [5.000, 100.000]

- **Baixo** [5.000, 20.000]
- **Adequado** [21.000, 70.000]
- **Excedente** [71.000, 100.000]

Consequentemente, uma base inicial de regras foi construída, possibilitando o mapeamento das interações entre as variáveis. O processo de definição das variáveis e criação das regras foi realizado em conjunto pelos membros do grupo, com auxilio do GPT-4, objetivando garantir que ambos os sistemas seguissem a mesma lógica e estrutura de avaliação. Durante o processo, foram necessárias algumas intervenções quanto a definição das variáveis para possibilitar uma representação mais realista do problema, evitando o excesso de variáveis com representações muito subjetivas.

REGRAS FUZZY
1. SE Experiência da Equipe é **Baixa** E Complexidade Técnica é **Alta** ENTÃO Nível de Risco é **Alto**.
2. SE Nível de Entrosamento da Equipe é **Baixo** OU Rotatividade da Equipe é **Alta** ENTÃO Nível de Risco é **Crítico**.
3. SE Definição de Requisitos é **Mal Definida** E Mudanças nos Requisitos são **Altas** ENTÃO Nível de Risco é **Crítico**.
4. SE Complexidade Técnica é **Média** E Tamanho do Projeto é **Grande** ENTÃO Nível de Risco é **Alto**.
5. SE Prazo é **Curto** E Mudanças nos Requisitos são **Altas** ENTÃO Nível de Risco é **Crítico**.
6. SE Orçamento Disponível é **Baixo** E Prazo é **Curto**, ENTÃO Nível de Risco é **Alto**.
7. SE Comunicação é **Ruim** OU Impacto do Ambiente Externo é **Alto** ENTÃO Nível de Risco é Crítico.
8. SE Maturidade das Ferramentas é **Baixa** E Complexidade Técnica é **Alta** ENTÃO Nível de Risco é **Alto**.
9. SE Orçamento Disponível é **Adequado** E Prazo é **Moderado** E Comunicação é **Boa** ENTÃO Nível de Risco é **Moderado**.

10. SE `Definição de Requisitos` é **Bem Definida** E `Experiência da Equipe` é **Alta** E `Nível de Entrosamento da Equipe` é **Alto**, ENTÃO `Nível de Risco` é **Baixo**.
11. SE `Impacto do Ambiente Externo` é **Baixo** E `Tamanho do Projeto` é **Pequeno** E `Prazo` é **Longo** ENTÃO `Nível de Risco` é **Baixo**.
12. SE `Mudanças nos Requisitos` são **Baixas** E `Comunicação` é **Ótima** E `Orçamento Disponível` é **Excedente** ENTÃO `Nível de Risco` é **Baixo**.

Tabela 2. Base de regras *Fuzzy*

2.2. CONSTRUÇÃO DO MODELO *FUZZY*

Após a definição das variáveis e regras, a segunda etapa envolveu a construção do modelo fuzzy. Cada sistema foi projetado na linguagem *Python,* em conjunto da biblioteca *Scikit-Fuzzy*, com base nas variáveis e nas regras pré-definidas, aplicando a lógica fuzzy para realizar a avaliação do risco de projetos de software. O modelo foi responsável por processar as entradas (valores das variáveis) e gerar uma avaliação de risco que pudesse ser usada para diagnosticar potenciais problemas em um projeto.

A construção do modelo fuzzy foi realizada de forma individual para cada membro do grupo, permitindo que as duas LLMs (GPT-4 e Gemini) colaborassem em contextos diferentes, com o objetivo de comparar os desempenhos de ambas. O processo incluiu a implementação do sistema de inferência fuzzy, bem como a adaptação das variáveis e regras dentro do contexto de cada aplicação.

2.3. DESENVOLVIMENTO DA APLICAÇÃO

A terceira e última etapa foi dedicada ao desenvolvimento da aplicação final, que integra o modelo fuzzy para avaliar o risco dos projetos. A aplicação foi feita com o *microframework Flask* e criada para ser interativa, permitindo que os usuários inserissem dados sobre o projeto (como prazos, recursos, e complexidade) e, com base nesses dados, o sistema calculava o risco associado ao projeto.

Assim como na etapa anterior, o desenvolvimento da aplicação foi feito de maneira independente por cada membro do grupo, com a utilização das respectivas LLMs (GPT-4 e

Gemini). Essa separação permitiu que os resultados de cada LLM fossem analisados de forma isolada e comparados posteriormente. O desenvolvimento da aplicação incluiu tanto a construção da interface do usuário quanto a integração do modelo fuzzy ao sistema.

3. RESULTADOS

3.1. GPT-4 (OPENAI)

O **GPT-4**, modelo de última geração da OpenAI, foi utilizado no desenvolvimento do sistema de avaliação de risco de projetos de software. Como modelo pago, o GPT-4 oferece funcionalidades avançadas, como a capacidade de anexar documentos e links para websites como contexto adicional na geração textual. Para o desenvolvimento do modelo e aplicação, foi anexado ao chat um documento contendo as informações básicas sobre as variáveis, conceitos e conjunto de regras, bem como um contexto para a criação do código, informando linguagem, bibliotecas e estruturação da aplicação.

Os resultados foram imediatos e coesos, mas alguns problemas surgiram devido à necessidade de fornecer um contexto extenso, o que entrou em confronto com as limitações de tokens do modelo. Como resultado, o GPT-4 acabou alterando ou removendo algumas das regras durante o processo de desenvolvimento. Entretanto, o sistema final se mostrou funcional e bem estruturado, com as regras essenciais sendo corretamente implementadas e o modelo fuzzy operando conforme esperado.

3.1.1. VANTAGENS OBSERVADAS

- **Resultados Imediatos e Coesos:** O GPT-4 produziu resultados de maneira rápida e coerente, facilitando a construção do sistema fuzzy e a aplicação de regras. As respostas fornecidas foram consistentes, o que permitiu avançar de forma eficiente no desenvolvimento do projeto. Como exemplo, o modelo foi capaz de gerar corretamente uma configuração para teste, que funcionou perfeitamente no modelo.

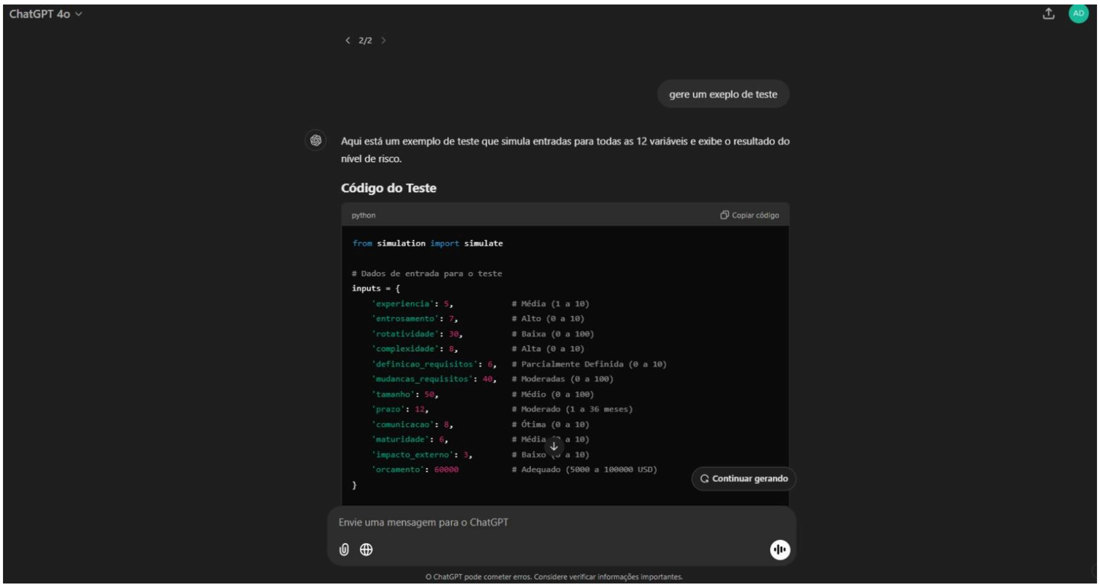


Figura 1. Geração correta de exemplos (GPT-4)

- **Auxílio na Correção de Erros:** Durante o processo, o GPT-4 demonstrou grande capacidade em identificar erros no código gerado, fornecendo sugestões úteis e precisas para correção. Isso foi essencial para resolver problemas técnicos rapidamente, garantindo o progresso contínuo do desenvolvimento.

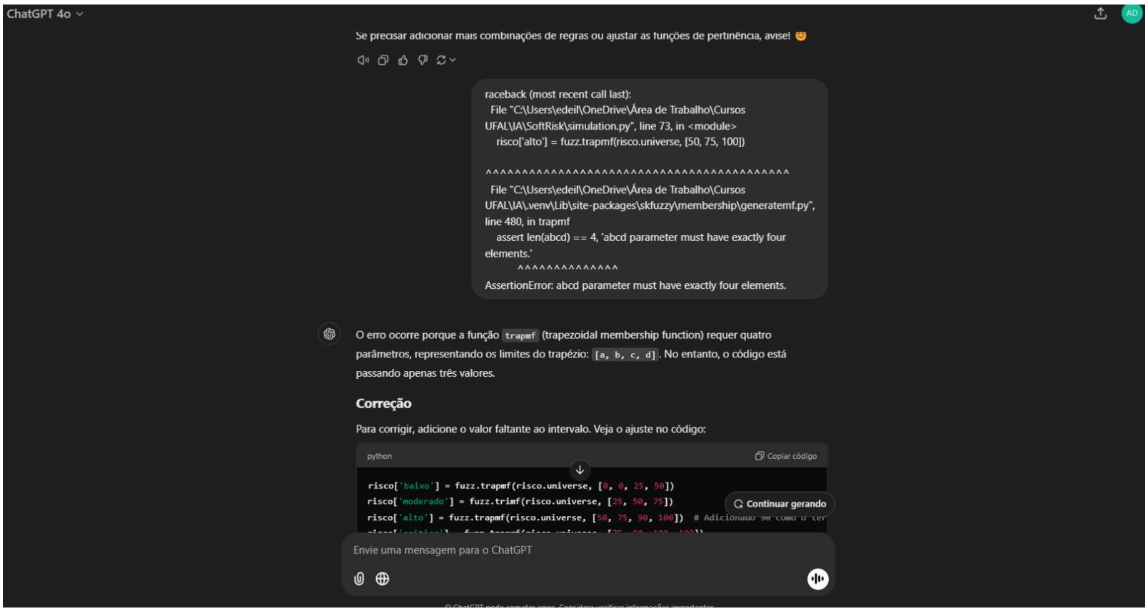


Figura 2. Solução para correção de erro na definição de função trapezoidal (GPT-4)

- **Compreensão de Imagens:** Uma das funcionalidades notáveis do GPT-4 foi sua habilidade em compreender imagens, o que se mostrou vantajoso para a documentação e visualização do sistema. Isso permitiu gerar representações gráficas claras e detalhadas dos resultados, facilitando a análise dos dados.

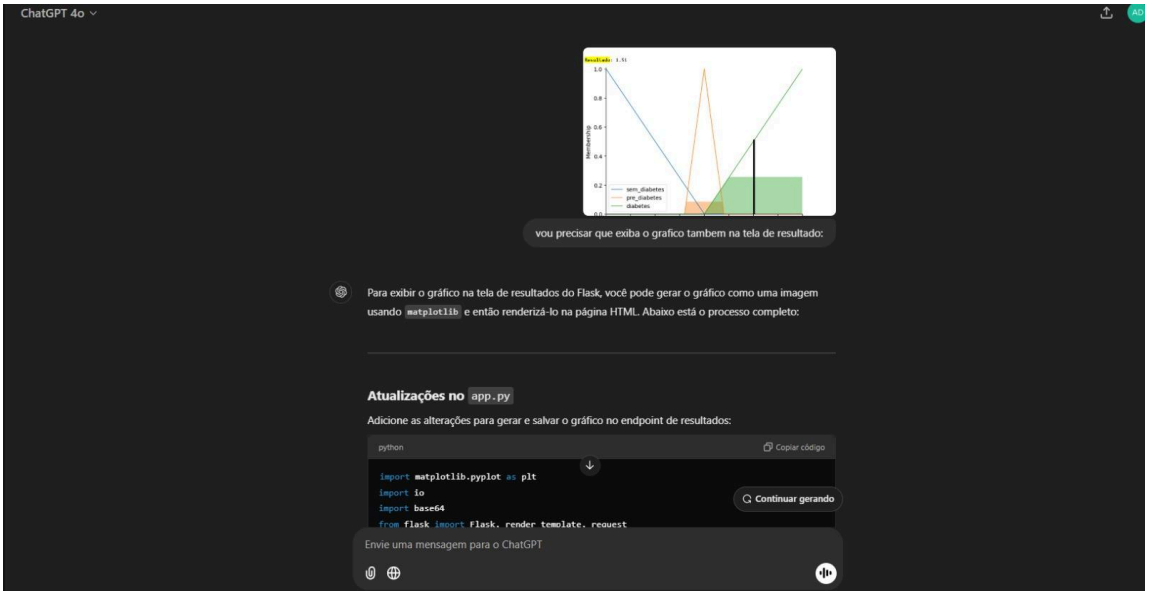


Figura 3. Geração de respostas com prompt de imagem e texto (GPT-4)

- **Geração de Documentação:** O GPT-4 também se destacou na geração de documentação, fornecendo resumos concisos e bem estruturados. A capacidade de criar documentação de forma rápida e eficaz ajudou a manter o projeto bem organizado e facilmente compreensível.

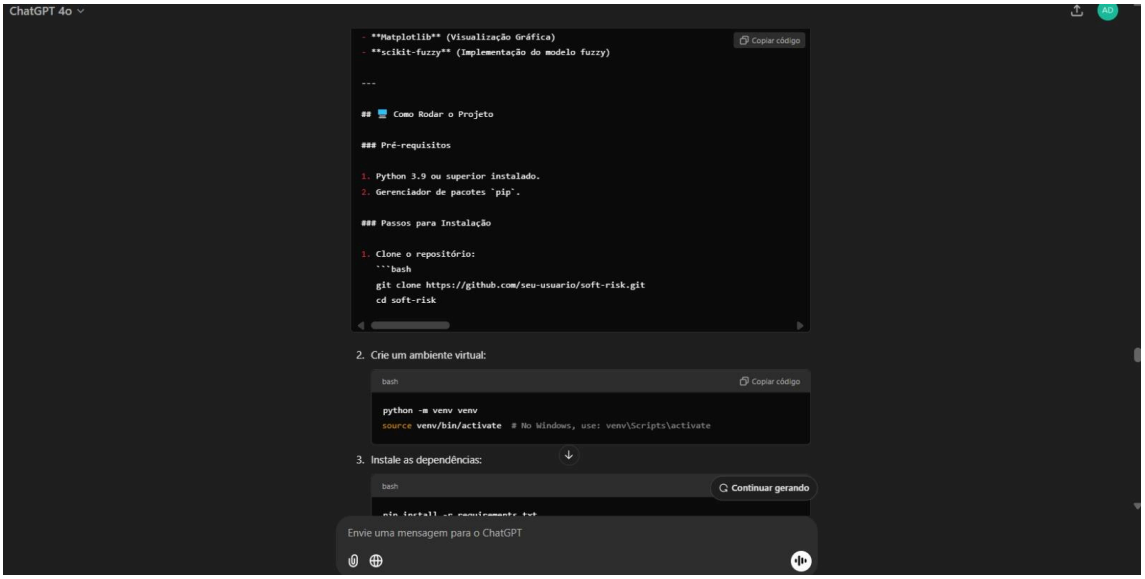


Figura 4. Geração de documentação para a aplicação (GPT-4)

3.1.2. DESAFIOS ENCONTRADOS

- **Erros de Importação de Bibliotecas:** Em algumas situações, o GPT-4 tentou executar o código que gerou, o que resultou em erros de importação de bibliotecas externas. Quando isso ocorreu, o modelo sugeriu soluções para que o usuário testasse localmente. Esse tipo de erro, embora comum, exigiu atenção adicional para garantir que o código fosse executado corretamente.

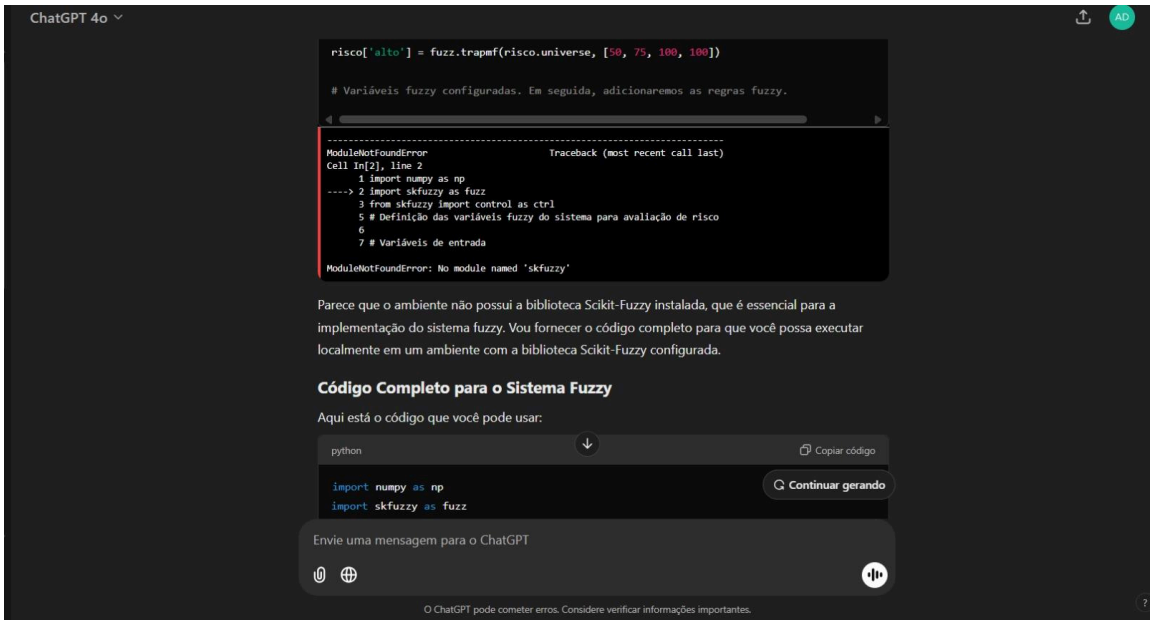


Figura 5. Erro na tentativa de execução de código (GPT-4)

- **Repetição de Instruções:** Em várias interações, o GPT-4 repetiu instruções relacionadas à instalação de dependências, mesmo que já tivessem sido instaladas em etapas anteriores do projeto. Essa repetição gerou um certo nível de redundância, o que exigiu uma maior vigilância por parte do desenvolvedor.

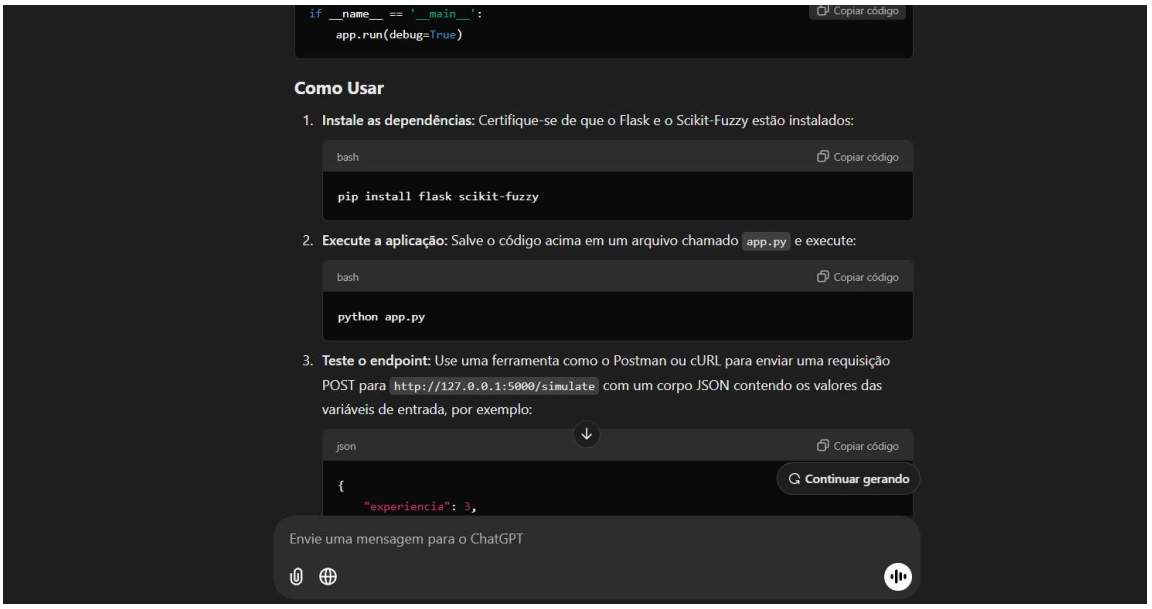


Figura 6. Repetição de instruções de instalação (GPT-4)

- **Perda de Contexto ao Longo do Processo:** Embora o GPT-4 facilite a construção de aplicações de forma modular, um desafio foi a perda gradual do contexto durante o desenvolvimento. À medida que o processo avançava, o modelo às vezes misturava diferentes partes do sistema, como a aplicação principal e o simulador, que haviam sido mantidas separadas no código. Esse tipo de inconsistência exigiu ajustes manuais para manter a organização do projeto.

3.1.3. RESULTADOS DA APLICAÇÃO

A aplicação segue o padrão de duas telas: uma para captura dos dados e outra que apresenta o resultado obtido pelo modelo fuzzy, juntamento com o gráfico de pertinência para a variável independente. A estilização foi feita através do *framework* de prototipação *TailwindCSS*.

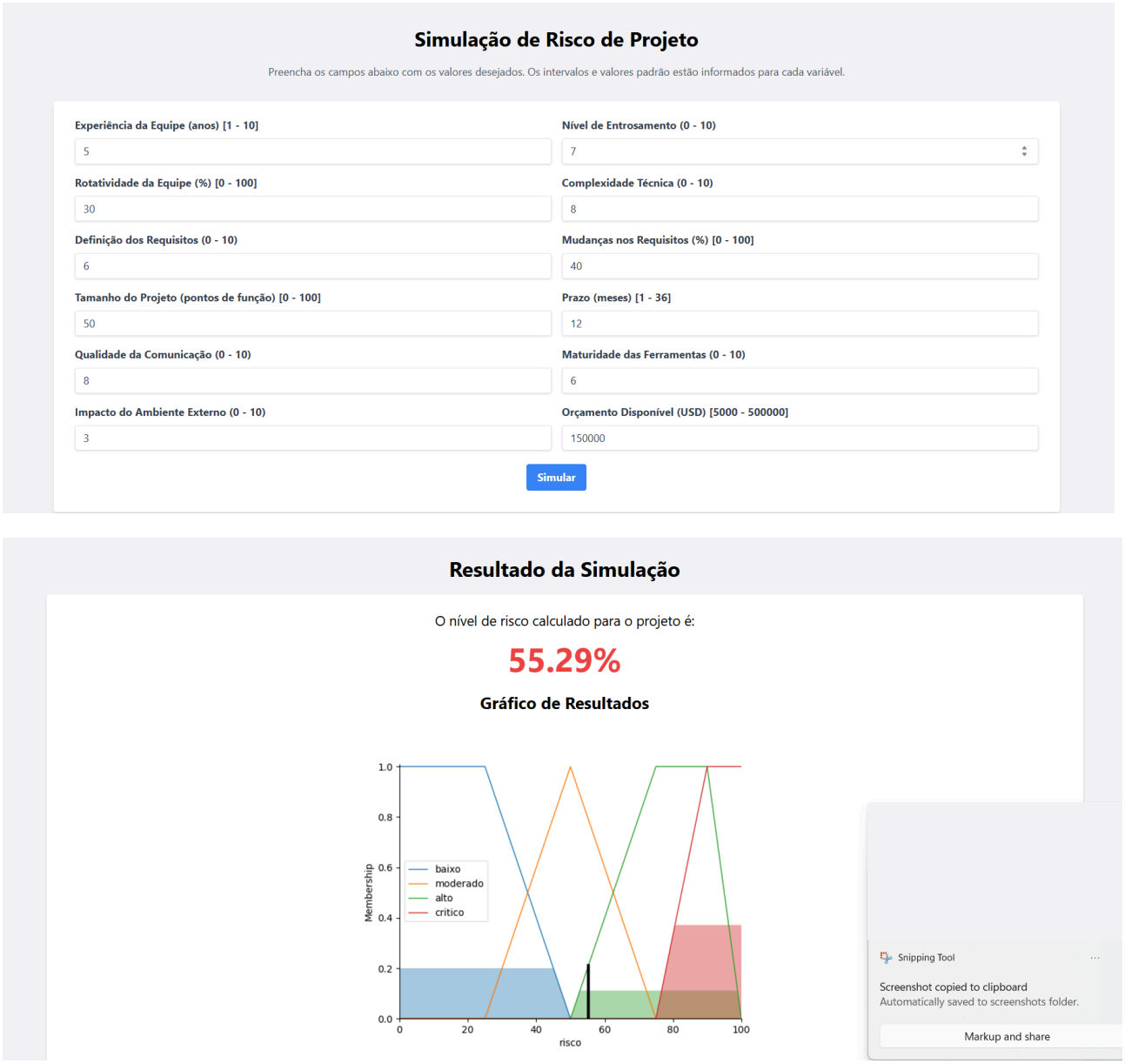


Figura 7. Telas da aplicação desenvolvida com GPT-4

3.2. GEMINI (GOOGLE)

O **Gemini 1.5 Flash**, versão gratuita da ferramenta de LLM desenvolvida pelo Google, foi utilizado para o desenvolvimento do segundo software de análise de risco. Embora o modelo tenha apresentado algumas limitações típicas da versão gratuita, como generalização excessiva nas respostas e erros ocasionais, ele se mostrou bastante funcional para tarefas de programação, oferecendo códigos organizados e coesos.

3.2.1. VANTAGENS OBSERVADAS

- **Descrição Clara das Etapas de Configuração:** O Gemini forneceu uma explicação detalhada sobre todas as etapas necessárias para a criação da base da aplicação Flask. O processo de criação do ambiente virtual, instalação das bibliotecas e a estruturação dos arquivos e diretórios foram bem descritos, o que facilitou a configuração inicial do sistema.

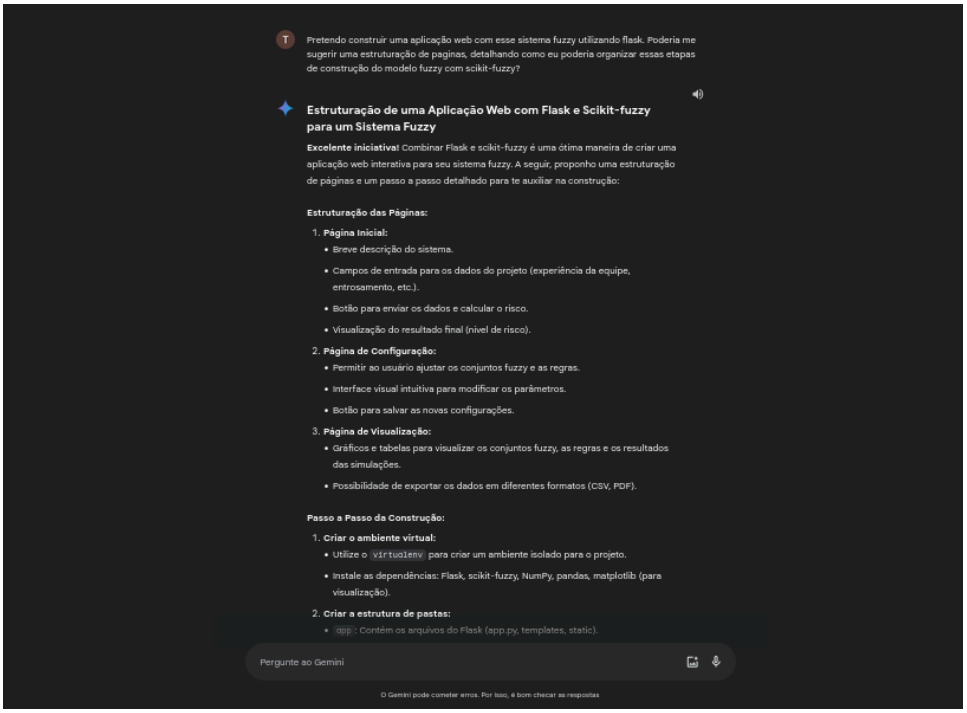


Figura 8. Descrição clara das etapas de construção de uma aplicação com *Flask* (Gemini)

- **Execução da Aplicação Bem Documentada:** O modelo forneceu uma boa descrição sobre o processo de execução da aplicação, detalhando de forma clara os passos necessários, incluindo a definição das variáveis de ambiente e a configuração dos componentes essenciais da aplicação.

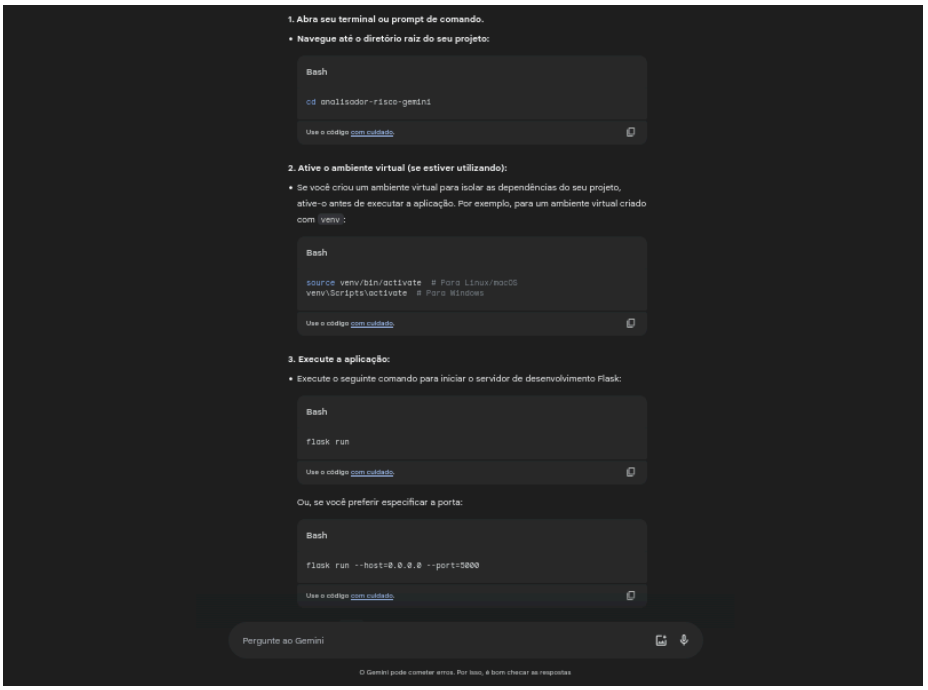


Figura 9. Documentação da execução da aplicação *Flask* (Gemini)

- **Geração de Código Coesa e Concisa:** O código gerado pelo Gemini foi bem estruturado e modular, seguindo os padrões da linguagem. Foi solicitada a criação de templates HTML com Jinja2 utilizando a abordagem de *blocks* para páginas e *macros* para generalização dos inputs, e o resultado foi surpreendente, apresentando uma estrutura clara e organizada que facilitou a construção do front-end.

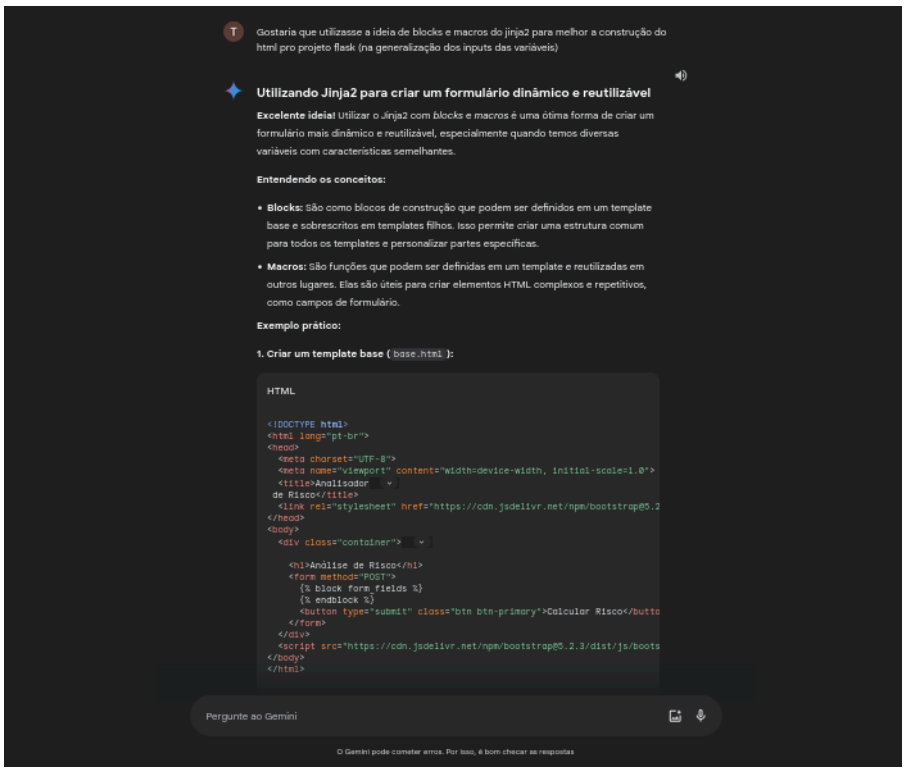


Figura 10. Geração de *block* e *macros* do *Jinja2* (Gemini)

- **Diagnóstico de Erros Eficaz:** O modelo demonstrou boa capacidade de diagnosticar erros, oferecendo diversas opções de explicação para os problemas encontrados, além de sugerir soluções viáveis. Embora algumas soluções não fossem totalmente adequadas, a maioria era coesa e útil para resolver os problemas.

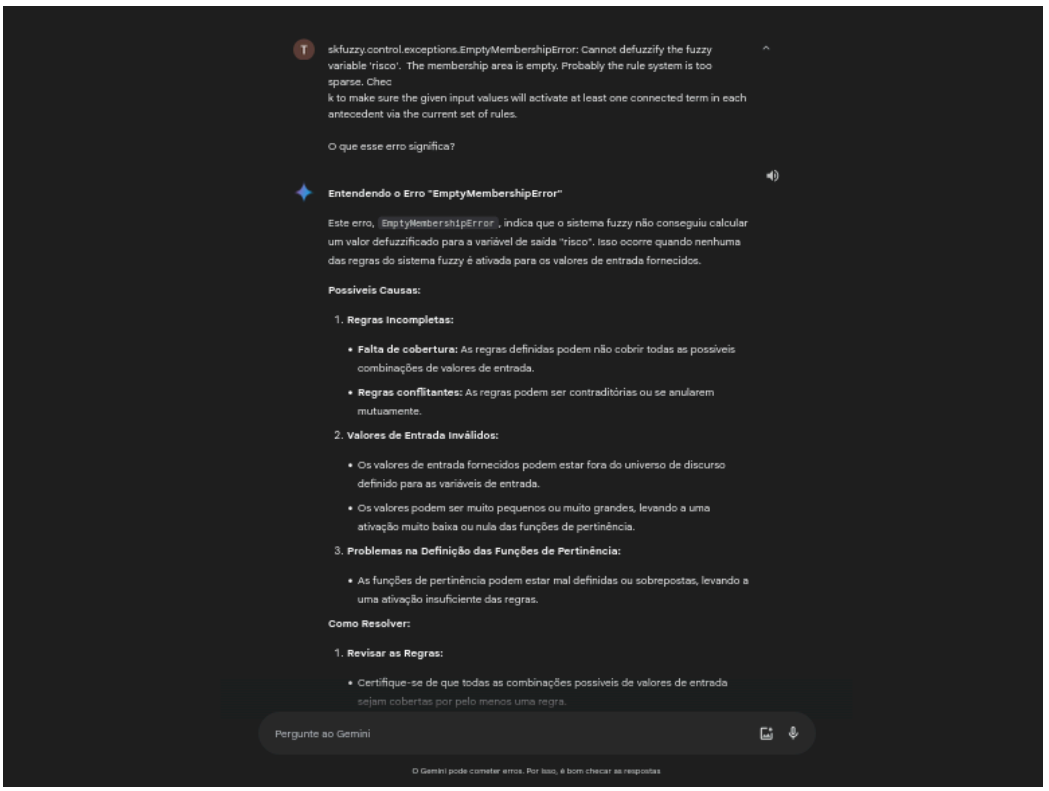


Figura 11. Diagnóstico para o erro de ausência de mapeamento de regras (Gemini)

3.2.2. DESAFIOS ENCONTRADOS

- **Respostas Generalistas:** Em alguns casos, as respostas fornecidas pelo Gemini foram excessivamente generalistas. Quando solicitado um detalhamento específico sobre uma parte do processo de construção do modelo com scikit-fuzzy, o modelo descreveu as etapas de forma genérica, sem o nível de profundidade necessário para a tarefa em questão.

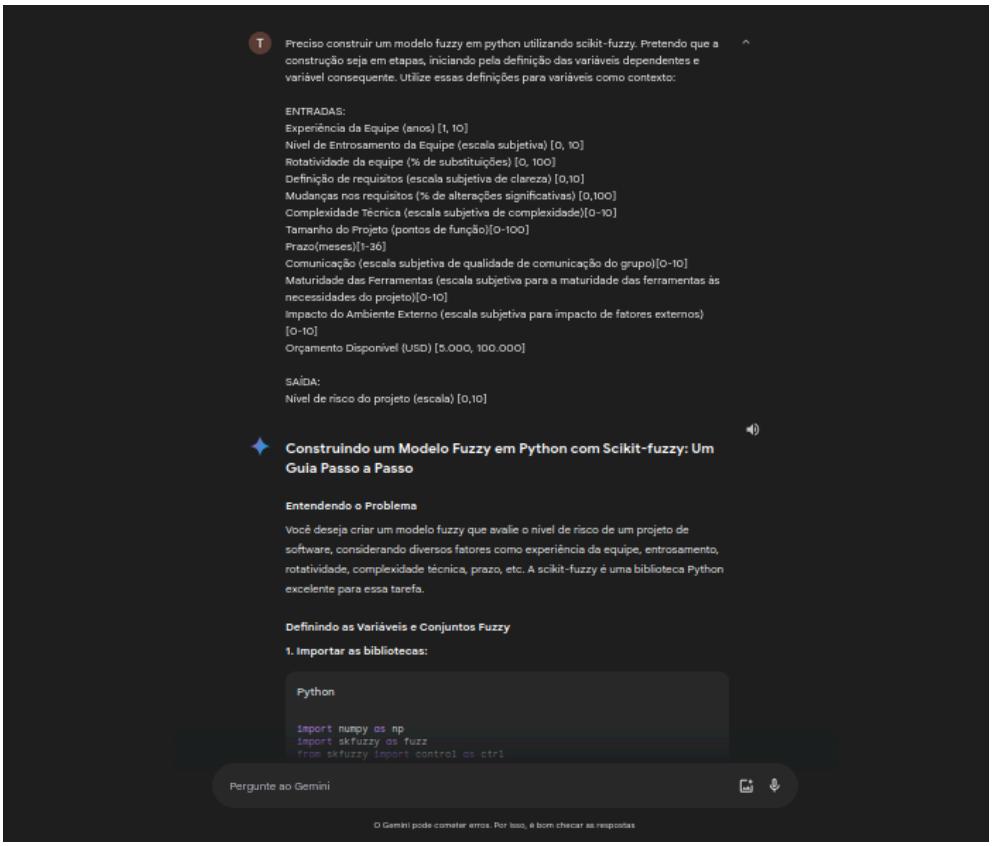


Figura 12. Resposta generalista para uma pergunta específica (Gemini)

- **Códigos Incompletos:** O Gemini gerou códigos incompletos em algumas situações, deixando espaços vazios para serem preenchidos. Esse problema foi notado principalmente na geração das funções de pertinência e regras utilizando o scikit-fuzzy. Foi necessário realizar um segundo prompt em ambos os casos para obter a resposta completa, visto que o processo de criação das regras era mais manual, exigindo a estruturação detalhada da biblioteca.

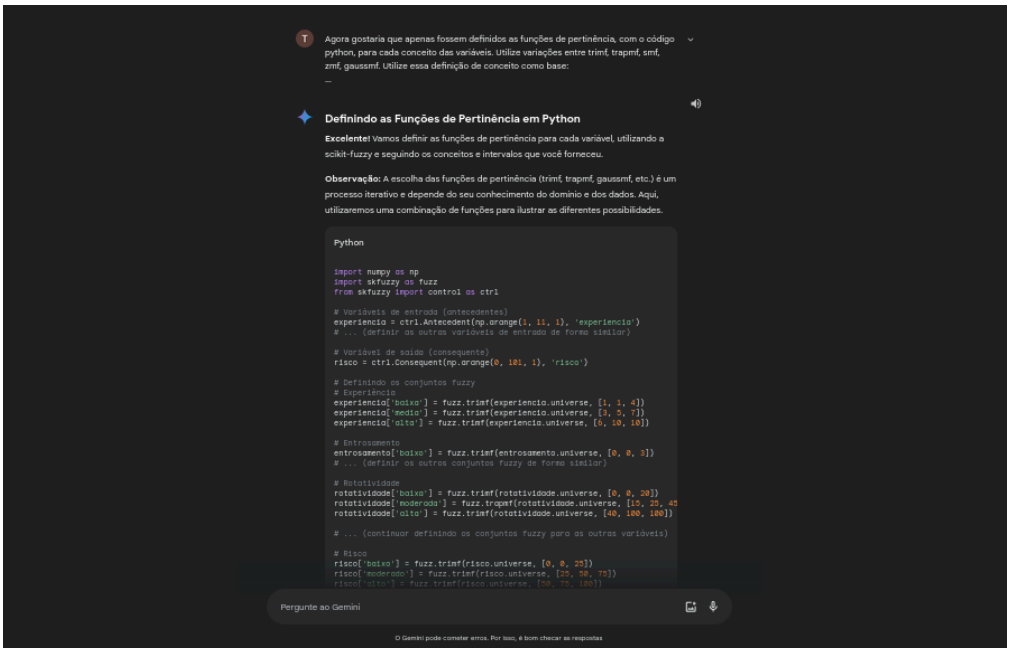


Figura 13. Presença de lacunas nos códigos (Gemini)

- **Perda de Contexto Rápida:** A perda de contexto foi um desafio frequente ao longo do desenvolvimento. Por exemplo, ao falar sobre a aplicação Flask e, em seguida, solicitar templates HTML com Bootstrap, o resultado gerado não levou em consideração uma das principais vantagens do Flask na geração de páginas server-side: o uso do Jinja2 para construção de templates, incluindo *blocks* e *macros*.

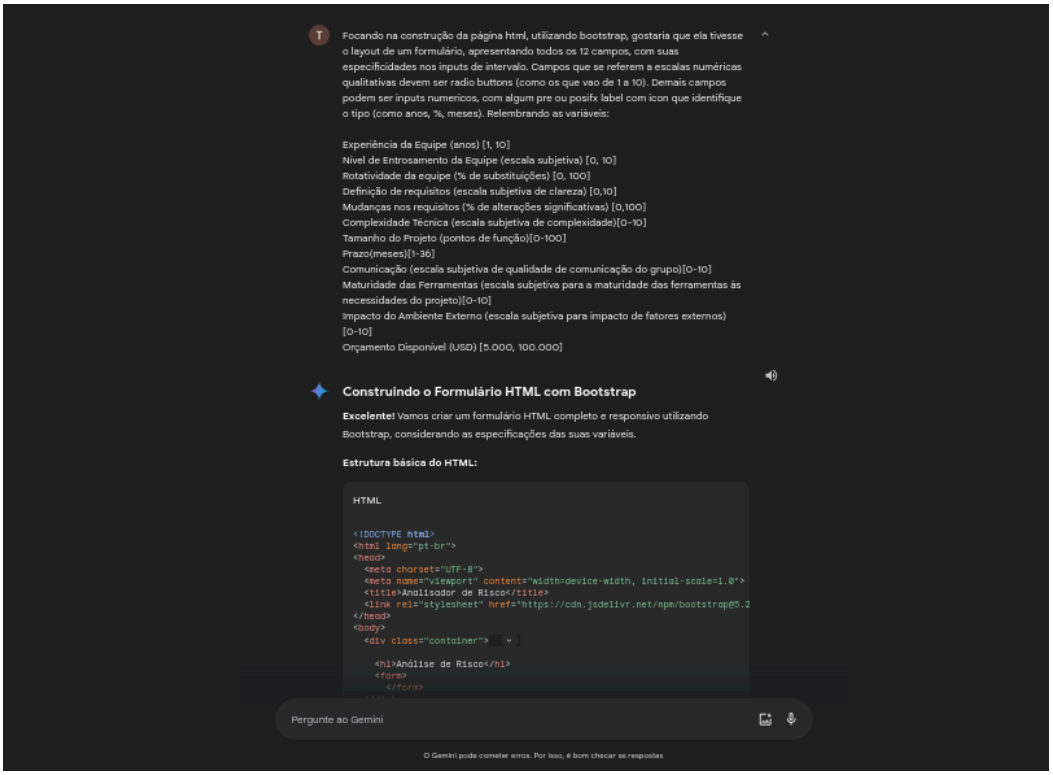


Figura 14. Perda de contexto na geração dos templates HTML da aplicação (Gemini)

3.1.1. RESULTADOS DA APLICAÇÃO

A aplicação segue o padrão de duas telas: uma para captura dos dados e outra que apresenta o resultado obtido pelo modelo fuzzy, juntamento com o gráfico de pertinência para a variável independente. A estilização foi feita através do *framework* de prototipação *Bootstrap* e os campos são personalizados de acordo com o tipo de entrada.

Figura 15. Telas da aplicação desenvolvida com Gemini

3.3. COMPARATIVO

Analisando ambos os LLMs, é possível observar uma consistência geral na geração de código, que é bem comentado, organizado e segue os padrões da linguagem. O **GPT-4**, por ser um modelo pago, destaca-se pela robustez e precisão de suas respostas, com uma menor incidência de erros. Suas saídas tendem a ser mais detalhadas, completas e baseadas em informações fundamentadas, apresentando menor tendência a introduzir conteúdo impreciso ou inventivo. Por outro lado, o **Gemini 1.5 Flash**, em sua versão gratuita, oferece respostas de qualidade aceitável, mas ocasionalmente mais generalistas e menos aderentes às especificidades das solicitações feitas. Apesar de exibir pequenos erros em alguns trechos de código, ele ainda se mostra uma ferramenta eficiente, especialmente para programadores com maior experiência, que podem corrigir ou ajustar rapidamente os detalhes imprecisos.

3.3.1. EXEMPLO DE RESOLUÇÃO DE PROBLEMAS

Um caso prático relevante foi o desenvolvimento da base de regras fuzzy. Nesse cenário, era crucial garantir que cada conceito pertencente às variáveis antecedentes fosse contemplado em pelo menos uma regra. A ausência desse mapeamento completo poderia levar a situações problemáticas, onde certos inputs não ativariam nenhuma regra. Isso causaria inconsistências no simulador fuzzy, como a impossibilidade de calcular a hiperárea (área total das funções de pertinência ativadas), levando a um erro crítico na divisão por zero ao tentar calcular o centro de gravidade.

```
skfuzzy.control.exceptions.EmptyMembershipError: Cannot defuzzify the fuzzy
variable 'risco'.  The membership area is empty. Probably the rule system is
too sparse. Check to make sure the given input values will activate at least
one connected term in each antecedent via the current set of rules.
```

O GPT-4 abordou esse problema de maneira eficaz e imediata. Ele utilizou a estrutura de tratamento de erros do Python para assegurar que valores incalculáveis não fossem submetidos ao processo de inferência fuzzy. Isso preveniu que a aplicação quebrasse quando encontrasse zonas de regras não ativadas. Essa solução resolveu a situação, mas não o problema como um todo, que acabou não sendo percebido até ser encontrado na outra aplicação.

```python
# --- Função de simulação ---
def simulate(inputs):
     try:
         for key, value in inputs.items():
```

```
            risk_simulation.input[key] = value  # Insere os valores
de entrada

        risk_simulation.compute()  # Executa o sistema fuzzy
        return round(risk_simulation.output["risco"], 2)  # Retorna o
resultado arredondado
    except KeyError as e:
        raise ValueError(f"Variável desconhecida: {e}")
```

No caso do Gemini, o problema não foi resolvido de forma tão imediata. A ausência de regras cobrindo todos os conceitos gerou um longo processo de investigação para entender a origem dos erros, que se manifestavam como *outputs* vazios no simulador fuzzy ao final da computação dos dados de entrada. Após identificar que o problema estava relacionado à falta de mapeamento de certos conceitos nas regras iniciais, foi solicitado ao modelo a geração de novas regras que contemplassem os conceitos negligenciados.

O Gemini conseguiu gerar essas novas regras com sucesso, corrigindo a lacuna inicial. No entanto, as regras geradas apresentaram pequenos erros, especialmente na nomeação de conceitos, que precisaram ser ajustados manualmente. Apesar disso, os ajustes necessários foram simples, e as novas regras resolveram o problema. Além disso, ampliaram significativamente a capacidade do modelo fuzzy de lidar com uma maior variedade de entradas, tornando-o mais adaptável e eficaz.

```
# Novas regras para cobrir os conceitos faltantes

# Experiência Média
rule13 = ctrl.Rule(experiencia['media'] &
complexidade_tecnica['media'], risco['moderado'])
rule14 = ctrl.Rule(experiencia['media'] & entrosamento['alto'],
risco['baixo'])

# Entrosamento Médio
rule15 = ctrl.Rule(entrosamento['medio'] & rotatividade['baixa'],
risco['baixo'])
rule16 = ctrl.Rule(entrosamento['medio'] &
complexidade_tecnica['baixa'], risco['baixo'])

# Rotatividade Baixa e Média
rule17 = ctrl.Rule(rotatividade['baixa'] &
```

```
definicao_requisitos['moderada'], risco['moderado'])
rule18 = ctrl.Rule(rotatividade['moderada'] &
mudancas_requisitos['baixas'], risco['baixo'])

#...
```

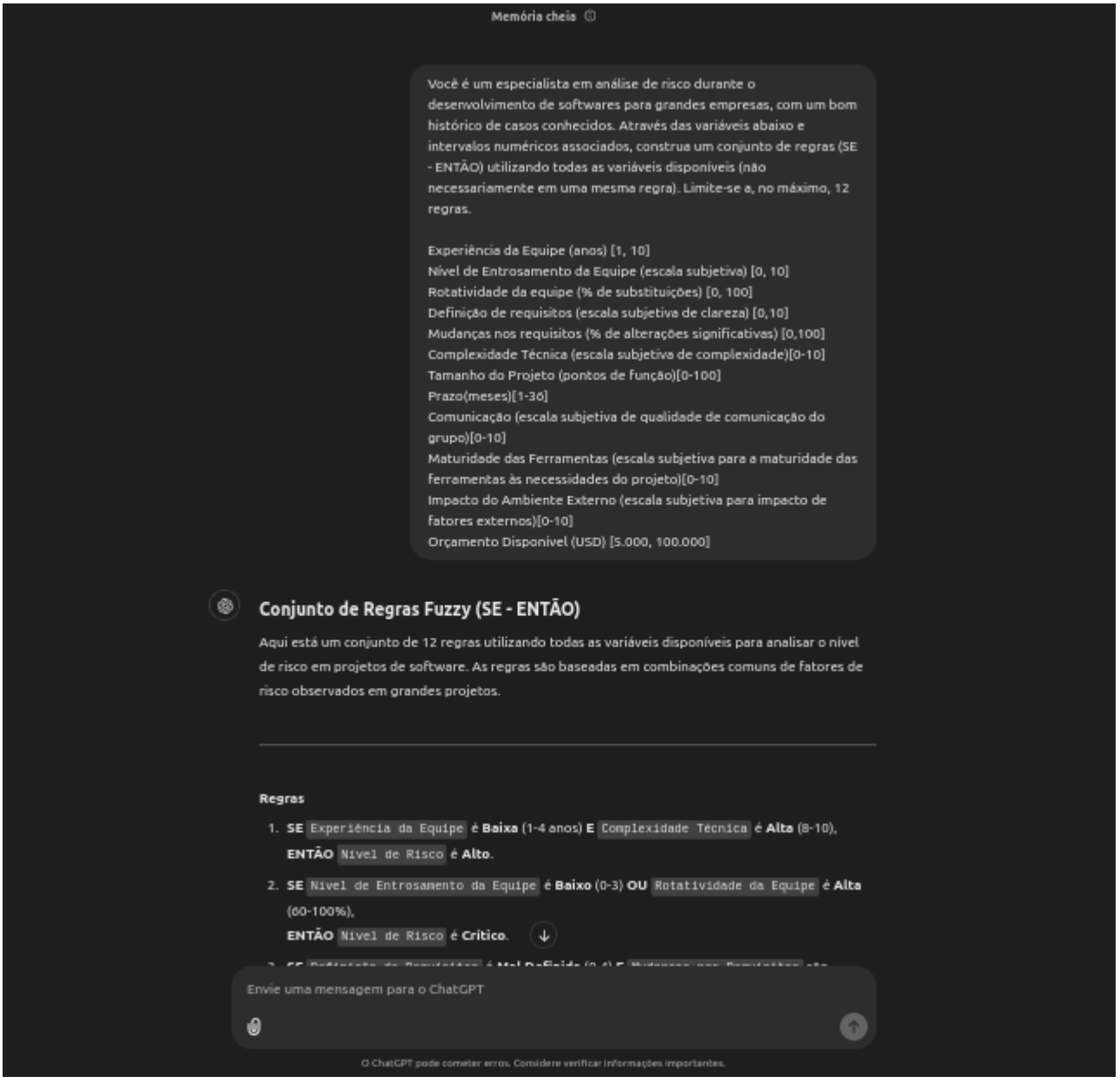
3.3.2. TABELA COMPARATIVA

CRITÉRIO	MELHOR LLM	CONCLUSÃO
Imediatismo na solução	GPT-4	Responde de forma precisa e rápida, utilizando estratégias robustas como tratamento de erros.
Detalhamento técnico	GPT-4	Produz respostas completas e bem fundamentadas, evitando lacunas ou omissões.
Consistência do código	GPT-4	Gera códigos mais coesos e organizados, com comentários claros e alinhados aos padrões da linguagem.
Flexibilidade na adaptação	Gemini	Adaptou-se bem à correção de problemas, ajustando regras e aprimorando o sistema fuzzy com orientações.
Qualidade na geração de regras fuzzy	GPT-4	Criou regras mais consistentes e precisas, com menor necessidade de ajustes manuais.
Diagnóstico de erros	GPT-4	Diagnóstico mais rápido e detalhado, com sugestões práticas e aplicáveis diretamente.
Custo-benefício	Gemini	Por ser gratuito, oferece uma solução viável com qualidade suficiente para tarefas de programação.
Manutenção do contexto	GPT-4	Conseguiu manter o contexto em interações mais longas, reduzindo a necessidade de reorientações.

Organização modular	GPT-4	Criou a aplicação de modo modularizado sem tanta intervenção do programador
Abertura a entradas variadas	Gemini	Após ajustes, gerou um sistema mais adaptável a uma gama maior de inp

Tabela 3. Comparativo entre os dois LLM's

ANEXO A: PROMPTS DE GERAÇÃO DOS CONCEITOS E REGRAS



Gostaria de que fosse definido, para cada uma das variáveis, uma divisão de conceitos fuzzy, informando os intervalos referentes a cada conceito. Retorne a informação das variáveis seguindo esse modelo:

- variável 1 (unidade) [Faixa]
  - conceito 1 [Faixa]
  - conceito 2 [Faixa]
  ...

Segue a definição das variáveis e seus respectivos conceitos fuzzy, com os intervalos definidos para cada conceito.

Variáveis e Conceitos Fuzzy

1. Experiência da Equipe (anos) [1, 10]

- **Baixa** [1, 4]
- **Média** [3, 7]
- **Alta** [6, 10]

2. Nível de Entrosamento da Equipe (escala subjetiva) [0, 10]

- **Baixo** [0, 3]
- **Médio** [2, 7]
- **Alto** [6, 10]

3. Rotatividade da Equipe (% de substituições) [0, 100]

- **Baixa** [0, 20]
- **Moderada** [15, 50]
- **Alta** [40, 100]

4. Definição de Requisitos (escala subjetiva de clareza) [0, 10]

- **Mal Definida** [0, 4]
- **Moderada** [3, 7]

Envie uma mensagem para o ChatGPT

O ChatGPT pode cometer erros. Considere verificar informações importantes.