Text-to-SQL com Sistemas Multiagentes

Artur Magalhães dos Santos*, Daniel Ferreira Schulz*,
Guilherme Narciso Lee*, Pedro Rezende Mendonça*
*Departamento de Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo, Brasil
{artur_santos, danielschulz, narcisolee2002, pedro.rezende.mendonca}@usp.br

***Abstract*—Efficient access to large volumes of structured data remains a challenge, especially for non-technical users. Text-to-SQL approaches aim to translate natural language questions into SQL queries, and recent advances in Large Language Models (LLMs) have improved their performance. However, most current solutions use single-agent architectures, which may limit modularity and validation.**

This paper investigates whether multi-agent architectures can outperform single-agent systems in Text-to-SQL tasks. We developed Mimir, a system composed of four specialized agents using the CrewAI framework, and compared it to a single-agent baseline. Experiments were conducted on two databases with 27 natural language questions and gold-standard SQL queries.

The results indicate that the multi-agent architecture achieves comparable accuracy to the single-agent model in manually audited queries, although at a higher computational cost. We observed that most discrepancies between expected and actual outputs were due to lexical mismatches (e.g., singular/plural forms and language translation), rather than logical errors in query construction. Furthermore, automated LLM auditing yielded a high false negative rate, highlighting the need for execution-based evaluation metrics. These findings suggest that while multi-agent systems offer modularity and extensibility, their benefits are not evident for low-complexity queries. Future work includes expanding the evaluation to more complex benchmarks, refining prompt engineering strategies, and exploring alternative agent topologies.

***Index Terms*—Text-to-SQL, Multi-agent Systems, LLM-as-a-Judge**

I. Introdução

O consumo eficiente a grandes volumes de dados estruturados é um desafio recorrente para aqueles que dependem da tomada de decisão orientada por dados. Embora a linguagem de consulta SQL seja amplamente adotada, sua utilização depende de conhecimento técnico especializado, restringindo o acesso a dados por parte de usuários não técnicos. Esse cenário motivou o desenvolvimento de abordagens Text-to-SQL, que visam traduzir perguntas em linguagem natural para consultas SQL.

Com o avanço dos modelos de aprendizado profundo e, mais recentemente, de Grandes Modelos de Linguagem (LLMs), Text-to-SQL tem obtido resultados melhores, inclusive em cenários de múltiplos domínios e esquemas complexos. No entanto, a maioria das soluções atuais utiliza abordagens com agentes únicos, generalizando a resolução dos problemas envolvidos na tarefa de tradução.

Diante disso, este trabalho busca responder à seguinte questão de pesquisa: abordagens baseadas em múltiplos agentes podem superar arquiteturas com agente único na tarefa de geração de consultas SQL a partir de linguagem natural?

Para investigar essa questão, este trabalho tem como objetivo comparar duas arquiteturas distintas para a tarefa de Text-to-SQL: uma baseada em um único agente, que centraliza todas as decisões, e outra composta por múltiplos agentes especializados que interagem para propor, revisar e refinar a consulta gerada. O estudo é conduzido por meio de um experimento controlado, considerando tanto a qualidade sintática das consultas geradas quanto a correção dos resultados obtidos após sua execução nos bancos de dados.

O restante deste artigo está organizado da seguinte forma: a Seção II apresenta os conceitos teóricos fundamentais relacionados a LLMs, sistemas multiagentes e Text-to-SQL. A seção III discute os trabalhos relacionados. A seção IV descreve o estudo de caso desenvolvido. A seção V apresenta os resultados e análise comparativa, e a seção VI conclui o trabalho e apresenta trabalhos futuros.

II. Fundamentação Teórica

A. Large Language Models

Os Grandes Modelos de Linguagem (do inglês LLM - Large Language Models) são modelos de Inteligência Artificial capazes de compreender e gerar linguagem natural [1]. De modo geral, utilizam distribuição de probabilidade condicional para prever a próxima palavra em uma sequência de texto [2]. Esses cálculos são realizados em redes neurais baseadas na arquitetura *Transformer*, que se apoia no mecanismo de atenção introduzido por Vaswani et al. [3]. Esse mecanismo permite a paralelização do treinamento e tratamento eficiente de dependências de longo alcance entre palavras.

Os avanços introduzidos pelo mecanismo de atenção possibilitaram a criação de modelos com um número significativamente maior de parâmetros. Essa escalabilidade é uma das principais características que diferenciam as LLMs de modelos de linguagem anteriores, permitindo que sejam treinadas com bilhões de parâmetros sobre vastas quantidades de dados textuais [4]. Ao fornecer exemplos e instruções diretamente na entrada, esses modelos passam a adaptar suas respostas a tarefas específicas, demonstrando a capacidade de aprendizado em contexto, conhecida como *in-context learning* [5].

Adicionalmente, técnicas como *Retrieval-Augmented Generation* (RAG) [6] têm sido empregadas para ampliar o escopo de atuação dos LLMs, ao permitir a recuperação de informações externas e atualizadas que complementam seu conhecimento treinado. Essa abordagem potencializa a aplicação dos modelos em tarefas que demandam conhecimento factual e especializado [7]. Com isso, torna-se possível

adaptar LLMs a diferentes domínios da atividade humana, como medicina, finanças, pesquisa acadêmica, desenvolvimento de software e tradução automática, entre outros [8].

B. Sistemas Multiagentes

Para aprimorar o desempenho dos LLMs, diversas técnicas e arquiteturas têm sido exploradas. Entre elas, destacam-se a cadeia de pensamento (*chain-of-thought*, [29]), engenharia de prompts, auto-refinamento e ReAct (*Reasoning and Acting*, [34]), que buscam fortalecer habilidades como raciocínio, planejamento e uso de ferramentas externas pelos modelos. Uma abordagem complementar que vem ganhando destaque é o uso de sistemas multiagentes, que têm se mostrado promissores na melhoria da factualidade e da capacidade de raciocínio dos modelos em comparação a estratégias baseadas em modelos isolados. Em alguns cenários, essas configurações multiagentes apresentam desempenho superior [9]. Nessa configuração, múltiplas instâncias do modelo interagem por meio de propostas, críticas e refinamentos mútuos ao longo de várias rodadas, com o objetivo de convergir para uma resposta mais confiável e precisa.

No contexto da Inteligência Artificial, um agente é definido como uma entidade computacional autônoma capaz de perceber seu ambiente, tomar decisões e executar ações de forma orientada a objetivos [10]. Já um sistema multiagente é composto por um conjunto desses agentes, que cooperam (ou competem) para resolver problemas complexos que ultrapassam a capacidade de agentes individuais [11].

As interações entre múltiplos agentes podem ser classificadas em três categorias principais: cooperativas, nas quais os agentes colaboram para atingir um objetivo comum; adversárias, em que cada agente busca maximizar seu próprio benefício; e mistas, que combinam elementos das duas abordagens [12]. Essa flexibilidade torna os sistemas multiagentes aplicáveis a uma ampla gama de domínios, como desenvolvimento de software, automação industrial, pesquisa científica, jogos e simulações complexas.

C. Text-to-SQL

O consumo e a análise de dados têm crescido nos últimos anos devido à valorização desse ativo estratégico. Como muitos desses dados estão estruturados em grandes bancos, dentro de uma estrutura relacional e frequentemente compostos por diversas tabelas, recuperar informações de forma precisa pode representar um desafio. Nesse contexto, Text-to-SQL dedica-se a democratizar o acesso a dados por meio de consultas em sistemas gerenciadores de bancos de dados [13].

De forma simples, Text-to-SQL é o processo de tradução de uma pergunta em linguagem natural humana para uma consulta SQL. Para isso, a abordagem deve compreender a intenção semântica da pergunta, identificar as tabelas e colunas, além de realizar operações de agregação e filtragem [14]. Por fim, o algoritmo pode ser avaliado sob diferentes aspectos, como o resultado final da consulta no banco de dados ou a similaridade da consulta em relação a sua correspondência com uma consulta de referência anotada manualmente.

Essa abordagem é útil em aplicações como sistemas conversacionais, ferramentas de análises de dados e em sistemas gerenciadores de banco de dados. No entanto, alguns aspectos ainda são desafios para essa área, como a ordenação dos predicados na linguagem SQL, complexidade de múltiplos domínios de informação, ambiguidade semântica das consultas entre outros [15]. Além disso, o desempenho dos modelos pode variar dependendo do esquema, forma de anotação das perguntas e da disponibilidade de dados de treinamento diversificados.

III. Trabalhos Relacionados

A tarefa de Text-to-SQL tem sido amplamente explorada nas últimas décadas, impulsionada pela crescente necessidade de tornar dados acessíveis a usuários sem conhecimento técnico em SQL. Diversas abordagens foram propostas ao longo do tempo, desde modelos baseados em regras até os mais modernos grandes modelos de linguagem. Nesta seção, são apresentados trabalhos relevantes sobre o tema, com ênfase nas suas contribuições, limitações e impacto no avanço do estado da arte.

A. Abordagens clássicas de Text-to-SQL

Os primeiros trabalhos de tradução entre linguagem natural e bancos de dados relacionais foram inspirados por técnicas da área de recuperação da informação (IR). Ferramentas como o DISCOVER [27], IR-style [28] e SPARK [24] implementaram mecanismos de busca por palavras-chave em bancos de dados relacionais. Essas abordagens tentavam identificar os elementos mais relevantes das consultas com base na correspondência textual entre as palavras-chave da pergunta e os elementos do esquema relacional, como nomes de tabelas e atributos. Embora úteis, tais métodos eram limitados na sua capacidade de representar intenções mais complexas, como agregações, junções ou condições compostas.

Avanços posteriores buscaram estruturar a interpretação da linguagem natural por meio de técnicas de *parsing* semântico. Sistemas como SQLizer [22], que sintetiza consultas a partir da linguagem natural, e o trabalho de Li e Jagadish [25], que propõe interfaces interativas para usuários, ilustram essa linha de pesquisa. Também se destaca a proposta de Iyer et al. [26], que utiliza retorno do usuário para refinar o mapeamento semântico, antecipando técnicas supervisionadas mais modernas.

Modelos baseados em aprendizado de máquina iniciaram um grande avanço ao substituir heurísticas rígidas por técnicas supervisionadas. O Seq2SQL [20] foi um dos primeiros a explorar o uso de aprendizado por reforço para gerar consultas SQL a partir de linguagem natural, considerando a correspondência com os resultados esperados no banco de dados. Posteriormente, Guo et al. [21] propuseram o uso de representações intermediárias para lidar com consultas mais complexas em ambientes multi-esquema. Já o RAT-SQL [23] introduziu um mecanismo de codificação relacional com atenção contextualizada, permitindo ao modelo incorporar de

forma mais eficaz a estrutura do banco de dados durante o processo de geração.

A consolidação dessas abordagens foi impulsionada pelo surgimento de *benchmarks* amplamente adotados na literatura. O WikiSQL [20] ofereceu um dos primeiros conjuntos em larga escala com pares pergunta-SQL, focando em consultas simples sobre tabelas únicas. Em contrapartida, o Spider [18] introduziu um *benchmark* de múltiplos domínios com consultas SQL complexas, múltiplas tabelas e diversos esquemas, estabelecendo um novo padrão para avaliação de generalização e robustez dos modelos. Esses *benchmarks* não apenas viabilizaram comparações consistentes entre trabalhos, mas também incentivaram o desenvolvimento de arquiteturas mais sofisticadas voltadas à compreensão semântica e à estrutura relacional dos dados.

B. Avanços com LLMs

A evolução dos *Language Models* para Text-to-SQL acompanhou os avanços gerais do processamento de linguagem natural (PLN). Inicialmente, modelos baseados em arquiteturas recorrentes (RNNs/LSTMs) e redes convolucionais mostraram-se limitados para capturar dependências de longo prazo e estruturas sintáticas complexas necessárias para a geração de consultas SQL [20].

A arquitetura *Transformer*, introduzida por Vaswani et al. [3], superou as limitações dos modelos precedentes ao adotar uma abordagem centrada em mecanismos de atenção. Essa inovação permitiu a captura de relações contextuais entre tokens independentemente de sua distância na sequência, graças ao mecanismo de *self-attention*. Além disso, a arquitetura possibilitou o processamento paralelo de sequências completas e a modelagem eficiente de hierarquias sintáticas complexas, características anteriormente inatingíveis com arquiteturas recorrentes.

Essa revolução introduzida com o desenvolvimento da arquitetura *Transformer* impulsionou o desenvolvimento dos Modelos de Linguagem modernos. Radford et al. [35] com o GPT estabeleceu um novo paradigma ao demonstrar três princípios fundamentais: o pré-treinamento em corpus massivos (centenas de GB de texto) possibilita a aprendizagem de padrões linguísticos profundos; o ajuste fino com poucos exemplos adapta eficientemente o modelo a tarefas específicas; e arquiteturas puramente auto-regressivas (*decoder-only*) mostram-se altamente eficazes para geração textual. Vale destacar que esses modelos operam de forma generativa, produzindo saídas coerentes a partir de prompts de entrada, característica que se tornou fundamental para aplicações práticas.

O sucesso dessa abordagem levou ao surgimento de sistemas como o ChatGPT e modelos especializados como o SQLCoder. Essas aplicações demonstram capacidade notável de gerar consultas SQL funcionais a partir de prompts em linguagem natural, seja incorporando informações do esquema do banco diretamente no contexto do prompt, seja através de ferramentas auxiliares que recuperam metadados da estrutura do banco de dados.

Leaderboard - Execution Accuracy (EX)

	Model	Code	Size	Oracle Knowledge	Dev (%)	Test (%)
	Human Performance *Data Engineers + DB Students*			✔		**92.96**
🏆1 Mar 11, 2025	AskData + GPT-4o *AT&T CDO - DSAIR* [Shkapenyuk et al. '25]		UNK	✔	75.36	**77.14**
🥈2 Apr 16, 2025	CHASE-SQL + Gemini *Google Cloud* [Pourreza et al. '24]		UNK	✔	74.90	76.02
🥉3 Feb 27, 2025	Contextual-SQL *Contextual AI*		UNK	✔	73.50	75.63
4 Dec 17, 2024	XiYan-SQL *Alibaba Cloud* [Yifu Liu et al. '24]	[link]	UNK	✔	73.34	75.63

Fig. 1: Benchmark BIRD comparando, o score de Execution Accuracy de agentes humanos e sistemas envolvendo modelos de linguagem

Assim, diversas abordagens utilizando LLMs como motor de inferência principal surgiram, cada uma com ferramentas de extração de dados de bancos, modelos específicos de linguagem e arquiteturas diferentes umas das outras, o que possibilitou a utilização dessas abordagens em diversos casos e variações de SQL diferentes.

Contudo, apesar desses avanços, os resultados em *benchmarks* como Spider [18] e BIRD [33] ainda revelam limitações significativas, particularmente em consultas complexas envolvendo múltiplas tabelas ou operações de agregação aninhadas, onde as taxas de acurácia frequentemente permanecem abaixo do patamar ideal para aplicações em produção.

C. Arquiteturas baseadas em Sistemas Multiagentes

Sistemas multiagentes podem ser construídos com diferentes topologias. Uma forma convencional é organizar os agentes de forma sequencial. Ou seja, um agente realiza um processamento, passando os resultados para o próximo, assim por diante.

Outra possibilidade é organizar os agentes em *mesh* (2). Na arquitetura em *mesh*, os agentes trabalham em colaboração, podendo ser todos conectados entre si. Um aspecto importante é que o controle não é delegado necessariamente a somente um agente, os agentes interagem entre si, com um objetivo comum .

A forma hierárquica de organização consiste um agente atuando como orquestrador, enquanto delega e escolhe outros agentes para realizar partes de uma tarefa principal. Aqui é notável a separação de responsabilidades. Os agentes chamados pelo orquestrador funcionam como funções a tarefas específicas, permitindo uma especialização detalhada de cada agente.

D. Justificativa

Apesar dos avanços descritos, grande parte dos trabalhos recentes depende de arquiteturas centralizadas com agentes

Fig. 2: Exemplo de sistema multiagentes *mesh*. Fonte: Strands SDK

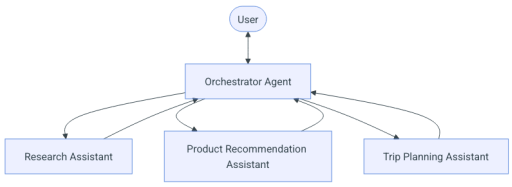

Fig. 3: Exemplo de sistema multiagentes *hierárquica*. Fonte: Strands SDK

únicos, o que pode limitar a capacidade de decomposição semântica e validação das consultas geradas. Neste contexto, abordagens multiagentes surgem como uma alternativa promissora para superar essas limitações, promovendo maior modularidade, especialização e robustez na geração de consultas SQL. Além disso, nenhum dos trabalhos citados realiza a comparação dos resultados entre modelos de agentes únicos com multiagentes, o que justifica a necessidade deste trabalho.

IV. Estudo de Caso

Como estudo de caso para Text-to-SQL, implementamos um sistema multi-agentes que tem como objetivo construir uma query em SQL a partir de um texto fornecido pelo usuário. Em muitos casos, usuários de sistemas de bancos de dados não possuem familiaridade com a linguagem SQL. Também, quando pensamos na elaboração de uma consulta, muitas vezes é mais fácil descrevê-la textualmente do que escrever diretamente a consulta em SQL.

Nosso sistema, denominado Mimir, constrói e executa uma consulta SQL fornecida pelo usuário, automaticamente selecionando quais tabelas e quais informações deve resgatar para que a consulta seja realizada com sucesso. Além do resultado final, para fins didáticos, ele também mostra qual foi a consulta executada.

A. Arquitetura

A aplicação foi elaborada utilizando a biblioteca CrewAI. Ela facilita a construção de sistemas multi-agentes (1) com uma API padronizada para as interações entre agentes, (2) facilidade de integração com plataformas de LLMs como a OpenAI e também (3) separa a especificação dos agentes - e.g. objetivos e descrições - do código de aplicação.

Nosso sistema é constituído por 4 agentes sequenciais, cada um com responsabilidades e objetivos diferentes. Na nomenclatura padrão do CrewAI, para definirmos um agente, precisamos especificar: (1) *role*, o papel que o agente vai desempenhar; (2) *goal*, o objetivo do agente; (3) *backstory*, contexto adicional do papel que o agente deve desempenhar; (4) llm, qual LLM o agente vai utilizar.

Um agente também é associado a uma *task*. Pela definição do CrewAI, "as *Tasks* fornecem todos os detalhes necessários para a execução, como uma descrição, o agente responsável, as ferramentas exigidas e mais, facilitando uma ampla variedade de complexidades de ação". Ou seja, representam a tarefa que o agente vai desempenhar.

A figura 4 apresenta a arquitetura da aplicação Mimir e os modelos LLM utilizados para cada agente.

Nossa aplicação possui os agentes (1) *Rewriter*, (2) *SQL Writer*, (3) *Optimizer* e o *Auditor*. Cada agente é executado sequencialmente, na ordem descrita. O primeiro deles, o *Rewriter*, tem como objetivo reescrever a frase do usuário. O usuário pode escrever uma frase errada ou até mesmo adicionar um texto extra acidentalmente. A premissa do primeiro agente é corrigir isto. A ideia de um agente de rescrita também advém de uma futura evolução, deste agente permitir o uso de *guardrails* de segurança para a aplicação, ou seja, que avalie se a consulta é maliciosa ou perigosa antes de iniciar qualquer processamento. O modelo utilizado para este agente foi o GPT 4o, da OpenAI.

A segunda etapa é escrever a consulta SQL, feita pelo agente *SQL Writer*. Este agente utiliza uma LLM mais robusta, o modelo *o3-mini*, da OpenAI, um dos mais capazes da geração atual das LLMs focadas em *reasoning*. A escolha do modelo vem do fato de consultas possivelmente serem complexas e necessitarem de maior capacidade de raciocínio. Este agente possui 2 ferramentas: (1) ferramenta que lê quais são as tabelas disponíveis no banco e (2) uma ferramenta que acessa as características dos campos de uma tabela. Parte das decisões do agente envolve saber quais ferramentas utilizar para que consiga concluir sua tarefa. Cada chamada é realizada em até cinco tentativas, sendo a última a que corresponde à consulta SQL selecionada como melhor solução.

Após a geração da consulta, o agente *Optimizer* é responsável por garantir que a consulta gerada responda a pergunta inicial e re-ajusta a consulta gerada conforme necessário. Ele foi pensado para permitir uma futura evolução, de ser um agente especializado em melhorar a qualidade de consultas complexas que tenham maior sensibilidade a performance. Nos nossos testes e exemplos, as consultas envolvem no máximo 4 tabelas, mas em um cenário real, esse valor pode ser muito maior, envolver diferentes chaves, funções, janelamento etc. Por agora, seu papel é garantir uma consulta que seja de acordo com o objetivo do usuário.

Por fim, é chamado o agente *Auditor*, que tem como responsabilidade garantir que a consulta gerada é válida e executável. Ele checa o resultado da consulta do *Optimizer* e garante que a consulta é executável. Seu objetivo é remover eventuais erros de identação, pontuação, entre outros. Um exemplo de prompt pode ser encontrado no código 4.

Listing 1: Exemplo de prompt para SQL Writer

```
sql_writer:
  role: >
    SQL Data Analyst and Data Engineer
  goal: >
    Generate a SQL query for a given query written in
         free text.
    This is the user query: {query}
  backstory: >
    You are a skilled analyst with a background in
        data manipulation
    and data processing. You have a talent for
        identifying patterns
    and extracting meaningful insights from a free
        form text and mapping
    into a valid and executable SQL query.
    You are aware that people need results that are
        interpretable,
    so you prefer to return textual results instead
        of considering just ids for tables.
```

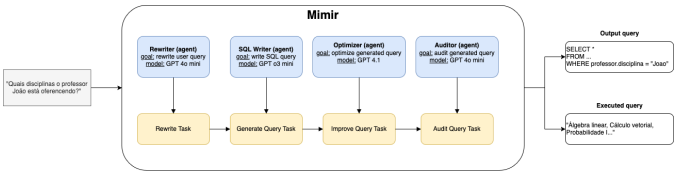

Fig. 4: Arquitetura da aplicação Mimir, desenvolvida para estudo de caso

Para mais informações sobre os prompts, cheque o Apêndice VII-A

B. Testes

Os testes foram realizados com dois bancos de dados pequenos, devido a limitação de recursos e ao número alto de chamadas de LLM causadas pela arquitetura multiagente. O primeiro, Cooperagri, possui quatro tabelas relacionadas a um circuito de produtores agrícolas e restaurantes consumidores de seus produtos (figura 5). O segundo, Alunos, possui quatro tabelas relacionadas a matrículas de alunos em cursos, ministrados por professores (figura 6). Esses bancos foram extraídos de exemplos construídos para a disciplina MAC5861 – Modelagem de Banco de Dados (2025, 1º semestre). Foram utilizadas 27 questões em linguagem natural extraídas das listas de exercícios da disciplina, e os gabaritos dos exercícios em consultas SQL foram adotados como padrão ouro (*ground truth*).

Lei *et al.* [19] propõem a mensuração da dificuldade de uma tarefa text-to-SQL a partir do número de espaços (tokens) na consulta SQL do padrão ouro. Utilizando essa métrica, comparamos a dificuldade do padrão ouro adotado neste artigo às dificuldades médias dos *benchmarks* Spider 1.0 e BIRD (figura 7). Observa-se que o padrão ouro adotado neste artigo tem dificuldade baixa, pouco acima da média do *benchmark* Spider 1.0 e abaixo da dificuldade média do *benchmark* BIRD. O *benchmark* Spider 2.0 foi desenvolvido especificamente para tarefas de alta dificuldade, adotando apenas consultas com

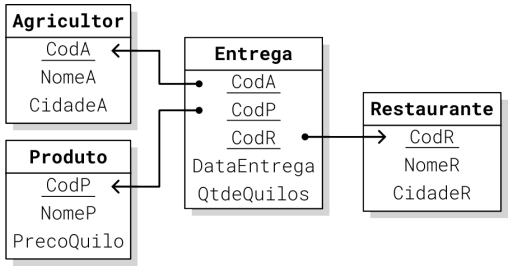

Fig. 5: Esquema do banco Cooperagri

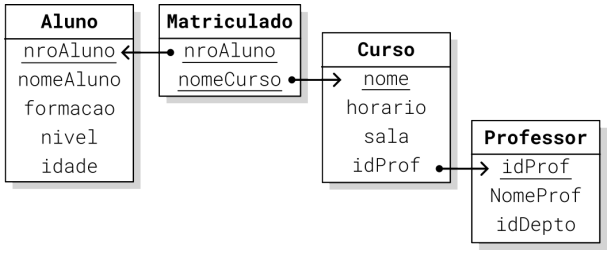

Fig. 6: Esquema do banco Alunos

mais de 50 tokens. A média de tokens na versão Spider 2.0-lite é 144,5, e 148,3 na versão completa.

Para avaliação dos resultados, formulamos métricas inspiradas nas metodologias do *benchmark* Spider 2.0-lite [19], e de Katsogiannis-Meimarakis & Koutrika [13].

O *benchmark* Spider 2.0-lite possui códigos de avaliação que consideram a acurácia de execução baseada na execução, ou seja, considera-se se todas as colunas das respostas do padrão ouro são preditas corretamente na resposta da consulta formulada por LLM. Assim, reduz-se o número de falsos negativos sem incremento do número de falsos positivos.

Já Katsogiannis-Meimarakis & Koutrika apresentam

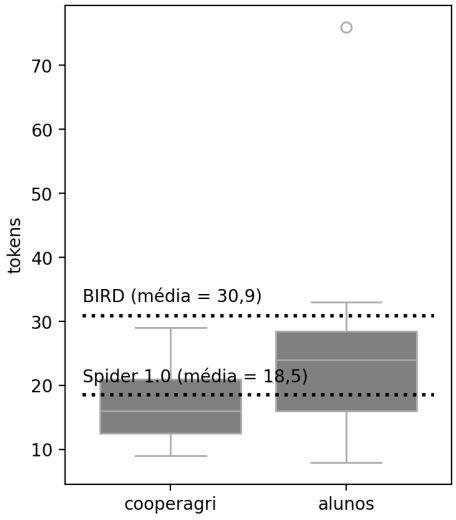

Fig. 7: Distribuição da dificuldade dos bancos Cooperagri e Alunos por número de tokens, em comparação às médias dos *benchmarks* BIRD e Spider 1.0

métricas para avaliação de aplicações text-to-SQL, que podem ser agrupadas como avaliação da consulta SQL gerada pela aplicação e avaliação dos resultados recuperados por essa consulta.

A partir dessa literatura, construímos duas métricas sintéticas:

1) Adequação da consulta SQL: valor booleano que avalia se a consulta gerada corresponde ao padrão ouro, ou seja, se possui a estrutura lógica esperada e retorna todas as colunas especificadas na cláusula SELECT do padrão ouro;
2) Adequação da resposta: valor booleano que avalia se a resposta da consulta gerada corresponde à resposta gerada pela consulta de padrão ouro, ou seja, se os valores das colunas estão corretos.

Além disso, a divisão mais ampla do *benchmark* Spider 2.0 nas versões lite e completa cobre dois tipos de tarefa text-to-SQL. A versão lite avalia o desempenho das LLMs para geração de consultas em SQL, tendo como dado de entrada apenas o esquema do banco de dados. Já a versão completa avalia a geração de uma resposta de texto complexa, que pode conter o resultado de várias consultas. Para isso, permite-se que a LLM acesse outros recursos para além do esquema, como acesso ao próprio banco de dados, a códigos auxiliares e outros documentos (figura 8).

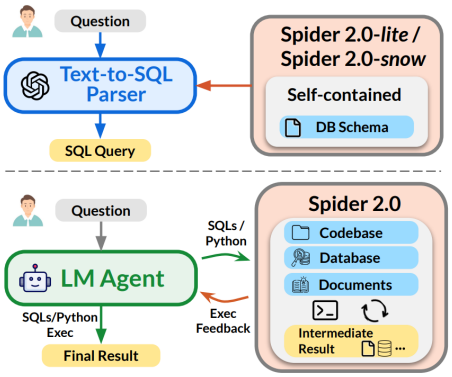

Fig. 8: Versões do Spider 2.0. Fonte: Lei *et al.* [19]

Partindo dessa divisão, propomos quatro testes para cada questão em linguagem natural:

1) Multiagente com acesso ao esquema e a ferramentas adicionais
2) Multiagente com acesso apenas ao esquema
3) Agente simples com acesso ao esquema e a ferramentas adicionais
4) Agente simples com acesso apenas ao esquema

Para calcular a pontuação de cada execução a partir das métricas propostas, realizamos uma avaliação por auditoria manual e outra utilizando a abordagem "LLM-as-a-Judge" [32], ou seja, a partir de uma nova chamada de LLM com um prompt que compara as consultas em SQL e as respostas ao padrão ouro. Como modelo para LLM-as-a-Judge, foi utilizado o GPT-3.5 Turbo, da OpenAI. As consultas e respostas adequadas receberam a pontuação 1 e as incorretas receberam a pontuação 0. Foram executadas 108 solicitações, com 486 chamadas a LLM, resultando em 16 pontuações. A figura 9 apresenta um diagrama simplificado da rotina de testes.

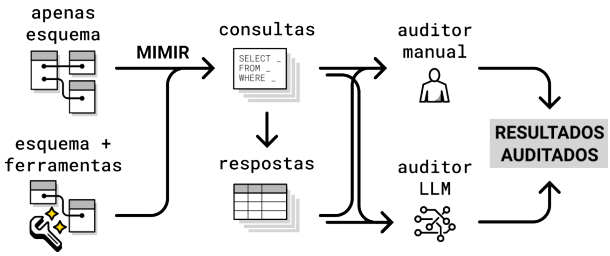

Fig. 9: Diagrama simplificado do fluxo de testes

V. Resultados e Discussão

As figuras 10, 11, 12 e 13 mostram a distribuição dos resultados por dificuldade da tarefa e o percentual geral de acertos (número de questões com pontuação 1 em relação ao total).

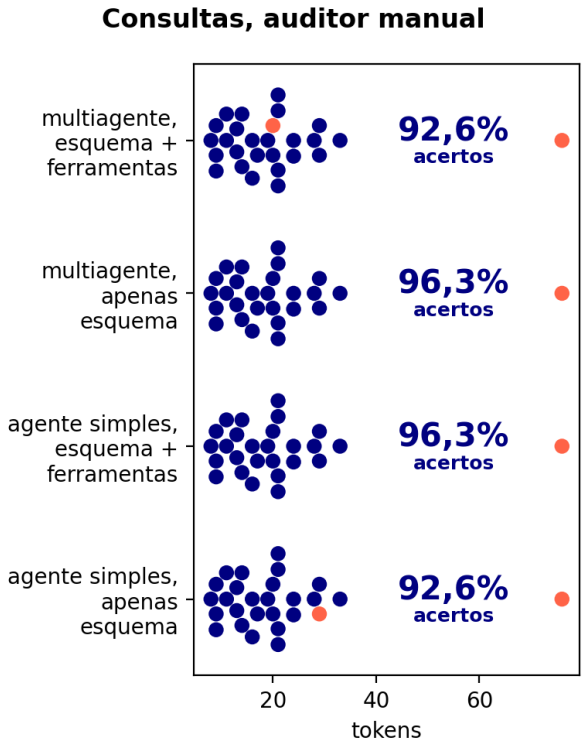

Fig. 10: Distribuição da avaliação das consultas em SQL por auditoria manual, por dificuldade da tarefa

No geral, observamos que a arquitetura multiagentes possui um desempenho próximo à arquitetura de agente simples na auditoria manual. Dado que o custo de uma solicitação em arquitetura multiagentes é multiplicado pelo número de agentes envolvidos, a arquitetura de agente simples mostra-se mais vantajosa para a faixa de dificuldade analisada neste artigo.

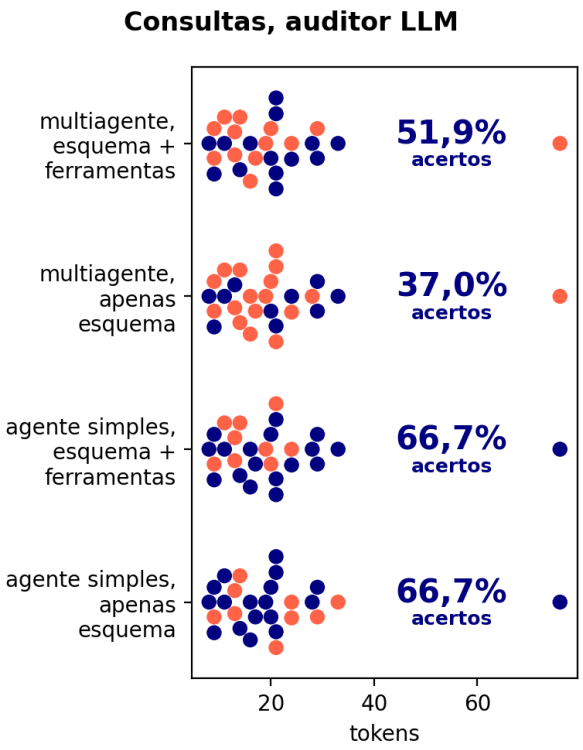

Fig. 11: Distribuição da avaliação das consultas em SQL por auditor LLM, por dificuldade da tarefa

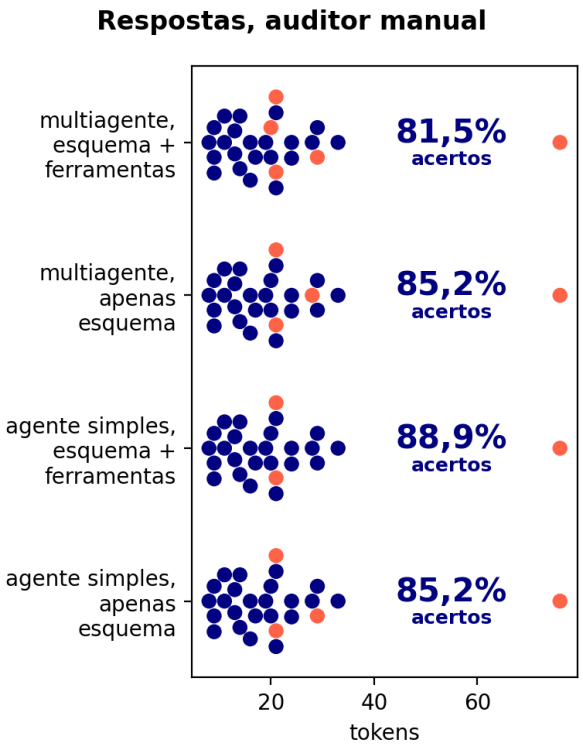

Fig. 12: Distribuição da avaliação das respostas por auditoria manual, por dificuldade da tarefa

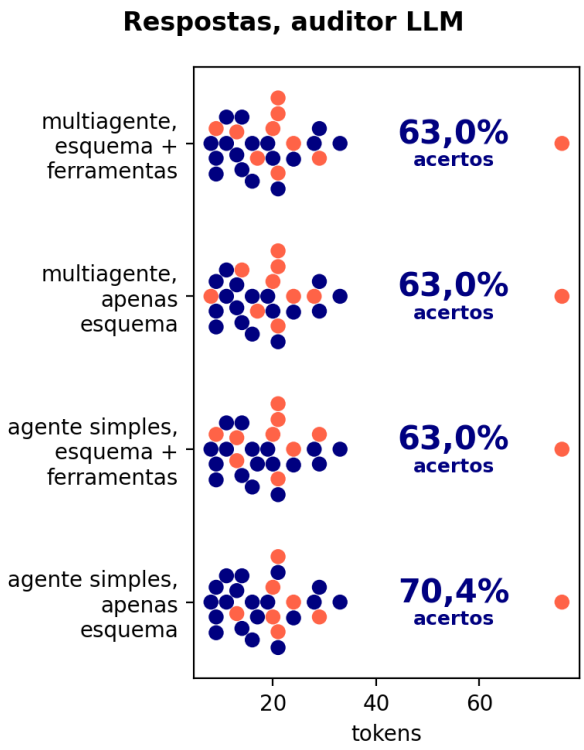

Fig. 13: Distribuição da avaliação das respostas por auditor LLM, por dificuldade da tarefa

Há uma diferença relevante entre a avaliação manual das consultas SQL e das suas respostas. A partir da leitura das consultas em SQL resultantes, identificamos que essa discrepância está relacionada majoritariamente ao uso de plural, tradução do português para o inglês e caracteres especiais. Por exemplo, a questão "Liste os códigos dos agricultores que já entregaram batatas, mas nunca entregaram cebolas" gera consultas com cláusulas WHERE que filtram por entradas com strings "cebolas" e "batatas" – palavras que são escritas no singular nas entradas do banco de dados. Os demais casos falharam ao traduzir cadeias de texto de filtros em português para inglês ou ao utilizar caracteres especiais nos rótulos das tabelas consultadas. Assim, a estrutura da consulta está logicamente correta e retorna as colunas do padrão ouro, mas os valores da tabela da resposta não correspondem ao padrão ouro. Por conta disso, a métrica equivalente à abordagem do *benchmark* Spider 2.0 é a relativa às respostas, por auditoria manual.

Ao comparar os resultados da auditoria manual – adotada como verdade base – ao auditor LLM, observamos que a auditoria LLM gera um número elevado de falsos negativos, ou seja, aponta consultas e respostas corretas como incorretas. O *recall* para a avaliação das consultas SQL é 56,8%, e 76,1% para as respostas, indicando que a comparação entre consultas em SQL é uma tarefa mais difícil. Contudo, há uma diferença relevante entre o desempenho do auditor LLM para solicitações de multiagente contra as solicitações de agente

simples, com melhor desempenho para as consultas SQL geradas por agente simples.

Por fim, observamos uma tendência de geração de consultas em SQL mais longas (figura 14). Também observamos que consultas geradas por LLM geram respostas com mais colunas do que o necessário. Isso reforça a abordagem de avaliação baseada na execução adotada pelo *benchmark* Spider 2.0 e adaptada neste artigo como mais adequada, pois considera se a resposta esperada está contida na resposta gerada e, com isso, reduz o número de falsos negativos.

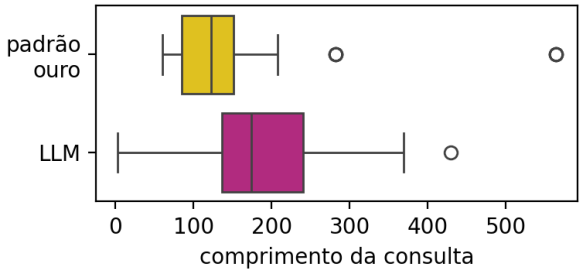

Fig. 14: Distribuição do comprimento em número de caracteres das consultas SQL no padrão ouro versus as consultas geradas por LLM

VI. Trabalhos Futuros

Uma avaliação mais definitiva sobre a vantagem da abordagem multiagentes exigiria o emprego de questões mais complexas e outros arranjos de papéis de agentes. Em trabalhos futuros, a aplicação implementada neste artigo deve ser avaliada com *benchmarks* completos, como o Spider 2.0.

Além disso, a melhoria dos prompts, tanto das *tasks* quanto dos agentes, utilizando técnicas de prompt engineering [30], [29], [31] potencialmente traria resultados melhores. Outro melhoria direta seria uniformizar o uso dos modelos com modelos de *reasoning*, como o *o3-mini*, já que estes possuem maior capacidade de raciocínio e performam melhor nos diferentes *benchmarks* de LLM. Outro caminho interessante seria testar diferentes topologias para o sistema multiagentes, como a topologia *mesh*, onde todos os agentes podem comunicar entre si, ou a topologia hierárquica, com um agente coordenando os demais.

References

[1] Y. Chang, X. Wang, J. Wang *et al.*, "A survey on evaluation of large language models," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 3, pp. 1–45, 2024.

[2] M. Shanahan, K. McDonell, and L. Reynolds, "Role play with large language models," *Nature*, vol. 623, no. 7987, pp. 493–498, 2023.

[3] A. Vaswani, N. Shazeer, N. Parmar *et al.*, "Attention is all you need," *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[4] M. A. K. Raiaan, M. S. H. Mukta, K. Fatema *et al.*, "A review on large language models: Architectures, applications, taxonomies, open issues and challenges," *IEEE Access*, vol. 12, pp. 26839–26874, 2024.

[5] T. Kojima, S. S. Gu, M. Reid *et al.*, "Large language models are zero-shot reasoners," in *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 22199–22213, 2022.

[6] P. Lewis, E. Perez, A. Piktus *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 9459–9474, 2020.

[7] W. Fan, Y. Ding, L. Ning *et al.*, "A survey on RAG meeting LLMs: Towards retrieval-augmented large language models," in *Proc. 30th ACM SIGKDD Conf. Knowl. Discov. Data Min.*, pp. 6491–6501, 2024.

[8] P. P. Ray, "ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope," *Internet Things Cyber-Phys. Syst.*, vol. 3, pp. 121–154, 2023.

[9] Y. Du, S. Li, A. Torralba *et al.*, "Improving factuality and reasoning in language models through multiagent debate," in *Proc. 41st Int. Conf. Mach. Learn. (ICML)*, 2023.

[10] Z. Xi, W. Chen, X. Guo *et al.*, "The rise and potential of large language model based agents: A survey," *Sci. China Inf. Sci.*, vol. 68, no. 2, pp. 121101, 2025.

[11] K. P. Sycara, "Multiagent systems," *AI Mag.*, vol. 19, no. 2, pp. 79, 1998.

[12] X. Li, S. Wang, S. Zeng *et al.*, "A survey on LLM-based multi-agent systems: Workflow, infrastructure, and challenges," *Vicinagearth*, vol. 1, no. 1, pp. 9, 2024.

[13] G. Katsogiannis-Meimarakis and G. Koutrika, "A survey on deep learning approaches for text-to-SQL," *VLDB J.*, vol. 32, no. 4, pp. 905–936, 2023.

[14] A. Wong, D. Joiner, C. Chiu *et al.*, "A survey of natural language processing implementation for data query systems," in *Proc. 2021 IEEE Int. Conf. Recent Adv. Syst. Sci. Eng. (RASSE)*, pp. 1–8, 2021.

[15] A. B. Kanburoğlu and F. B. Tek, "Text-to-SQL: A methodical review of challenges and models," *Turk. J. Electr. Eng. Comput. Sci.*, vol. 32, no. 3, pp. 403–419, 2024.

[16] G. Li, H. A. K. Hammoud, H. Itani *et al.*, "CAMEL: Communicative agents for 'mind' exploration of large language model society," in *Proc. 37th Int. Conf. Neural Inf. Process. Syst. (NeurIPS)*, pp. 51991–52008, 2023.

[17] W. Chen, Y. Su, J. Zuo *et al.*, "AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors," in *Proc. 12th Int. Conf. Learn. Represent. (ICLR)*, 2024.

[18] T. Yu, R. Zhang, K. Yang *et al.*, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," *arXiv preprint arXiv:1809.08887*, 2018.

[19] Lei, F., Chen, J., Ye, Y., Cao, R., Shin, D., Su, H.,*et al.*, "Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows", *arXiv preprint arXiv:2411.07763*, 2024.

[20] V. Zhong, C. Xiong, and R. Socher, "Seq2SQL: Generating structured queries from natural language using reinforcement learning," *arXiv preprint arXiv:1709.00103*, 2017.

[21] J. Guo, Z. Zhan, Y. Gao *et al.*, "Towards complex text-to-SQL in cross-domain database with intermediate representation," *arXiv preprint arXiv:1905.08205*, 2019.

[22] N. Yaghmazadeh, Y. Wang, I. Dillig *et al.*, "SQLizer: Query synthesis from natural language," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, pp. 1–26, 2017.

[23] B. Wang, R. Shin, X. Liu *et al.*, "RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguist. (ACL)*, pp. 7567–7578, 2020.

[24] Y. Luo, X. Lin, W. Wang, and X. Zhou, "SPARK: Top-k keyword query in relational databases," in *Proc. 2007 ACM SIGMOD Int. Conf. Manag. Data*, pp. 115–126, 2007.

[25] F. Li and H. V. Jagadish, "Constructing an interactive natural language interface for relational databases," *Proc. VLDB Endow.*, vol. 8, no. 1, pp. 73–84, 2014.

[26] S. Iyer, I. Konstas, A. Cheung *et al.*, "Learning a neural semantic parser from user feedback," *arXiv preprint arXiv:1704.08760*, 2017.

[27] V. Hristidis and Y. Papakonstantinou, "DISCOVER: Keyword search in relational databases," in *Proc. 28th Int. Conf. Very Large Databases (VLDB)*, pp. 670–681, 2002.

[28] V. Hristidis, Y. Papakonstantinou, and L. Gravano, "Efficient IR-style keyword search over relational databases," in *Proc. 2003 Int. Conf. Very Large Databases (VLDB)*, pp. 850–861, 2003.

[29] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS 2022)*, pages 1800–1813. Curran Associates Inc., Red Hook, NY, USA, 2022.

[30] Tom B. Brown, Benjamin Mann, Nick Ryder, et al. Language models are few-shot learners. In *Proceedings of the 34th International Conference*

on Neural Information Processing Systems (NeurIPS 2020), pages 159–183. Curran Associates Inc., Red Hook, NY, USA, 2020.

[31] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS 2023)*, pages 377–395. Curran Associates Inc., Red Hook, NY, USA, 2023.

[32] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, et al. Judging LLM-as-a-judge with MT-bench and Chatbot Arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS 2023)*, pages 2020–2048. Curran Associates Inc., Red Hook, NY, USA, 2023.

[33] Jinyang Li, Binyuan Hui, Ge Qu, et al. Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs. *Advances in Neural Information Processing Systems*, 36, 2024.

[34] Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv preprint arXiv:2210.03629, 2023.

[35] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. OpenAI Research, 2018. URL

VII. Apêndice

A. Prompts

Prompts utilizados nos agentes:

Listing 2: Exemplo de prompt para Rewriter

```
rewriter:
  role: >
    Rewrite and validate human user input text for {
        query}
  goal: >
    Rewrite a human user input text into a meaningful
         equivalent
    text removing any non-essential information and
        ensuring the
    text is clear, concise and respectful.
  backstory: >
    You are an experienced analyzer and rewriter with
         a talent for
    rewriting text that maintains content but it is
        more clear.
    You excel at organizing information in a clear
        and concise manner,
    making a text more readable and appropriate.
```

Listing 3: Exemplo de prompt para Optimizer

```
optimizer:
  role: >
    SQL Data Analyst that verifies the quality of a
        SQL query
  goal: >
    Verify if SQL query provided solves the problem
        described in the provided
    question. If not, correct the query.
  backstory: >
    You are a skilled analyst with a background in
        data interpretation
    and SQL queries. You are aware that people need
        results that are interpretable,
    so you prefer to return textual results instead
        of considering just ids for tables.
    You have a talent for identifying patterns and
        making sure the SQL solves the problem
    described in the provided question.
```

Listing 4: Exemplo de prompt para Auditor

```
auditor:
  role: >
    Audits SQL query
  goal: >
    Make sure a SQL query is valid and executable
  backstory: >
    You are a skilled analyst with a background in
        data interpretation
    and SQL. You are knowledgeable on SQL and
        databases.
```

Os demais prompts utilizados podem ser encontrados na URL: Github repo

B. Dados

Os dados utilizados para criação e testes se encontram na URL: Github repo