UAB
Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Planas Martínez, Aleix. *Avaluació de LLMs de codi obert en tasques de Text-to-SQL*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2025 (Grau en Enginyeria de Dades)

This version is available at https://ddd.uab.cat/record/337330

under the terms of the  license.

Avaluació de LLMs de codi obert en tasques de Text-to-SQL

Aleix Planas Martinez

Resum– Aquest treball té com a objectiu analitzar les capacitats dels models de llenguatge de gran escala (Large Language Models, LLMs) de codi obert en el context del problema Text-to-SQL, que consisteix en la traducció de consultes formulades en llenguatge natural a sentències SQL executables sobre una base de dades. Al llarg del projecte es duran a terme diferents experiments amb diversos models i configuracions, amb la finalitat d'identificar-ne les fortaleses, les limitacions i els factors que influeixen en el seu rendiment. A més, s'estudiaran diferents estratègies de construcció de prompts i de recuperació d'informació sobre l'esquema de la base de dades, amb l'objectiu de determinar quins mètodes permeten obtenir una major precisió i fiabilitat en la generació automàtica de consultes SQL. Els resultats obtinguts permetran valorar fins a quin punt aquestes tecnologies poden ser aplicades de manera efectiva en entorns reals d'accés i explotació de dades.

Paraules clau– Models de Llenguatge de Gran Escala (LLM), Text-to-SQL, prompt, Recuperació Augmentada per Generació (RAG).

Abstract– This project aims to analyze the capabilities of open-source Large Language Models (LLMs) in the context of the Text-to-SQL problem, which consists of translating natural language queries into executable SQL statements over a database. Throughout the project, a series of experiments will be conducted using different models and configurations in order to identify their strengths, limitations, and the factors that influence their performance. In addition, various prompt engineering and schema retrieval strategies will be evaluated to determine which approaches achieve the highest levels of accuracy and reliability in automatic SQL query generation. The results obtained will provide insights into the practical applicability of these technologies in real-world data access and management scenarios.

Keywords– Large Language Models (LLMs), Text-to-SQL, prompt, Retrieval-Augmented Generation (RAG).

1 Introducció - Context del treball

Aquest projecte s'emmarca en l'àmbit dels models de llenguatge de gran escala (Large Language Models, LLMs)[1] i la seva integració amb sistemes de gestió de dades. Els LLM són models d'intel·ligència artificial entrenats amb grans volums de text, capaços de comprendre i generar llenguatge natural, adaptar-se al context i resoldre diverses tasques lingüístiques mitjançant patrons estadístics apresos durant l'entrenament.

En els darrers anys, aquestes tecnologies han esdevingut una eina clau en l'automatització de processos i la interacció amb dades estructurades, tot i que encara presenten limitacions en la fiabilitat de les respostes i en la generació de consultes complexes sobre bases de dades [2].

En paral·lel, les bases de dades continuen sent un element fonamental en els sistemes informàtics, però l'accés a la informació requereix habitualment coneixements de SQL, fet que limita la seva utilització per part d'usuaris no tècnics.

La motivació d'aquest treball és estudiar el rendiment de models de llenguatge de codi obert aplicats a la generació de consultes SQL, analitzant diferents estratègies de generació i *prompt engineering* per millorar-ne la precisió.

Finalment, el projecte té com a objectiu el desenvolupament d'un sistema de xat offline per a la consulta de bases de dades en llenguatge natural, sense dependència de serveis externs ni coneixements de SQL per part de l'usuari.

• E-mail de contacte: aleixplanasmartinez@gmail.com
• Treball tutoritzat per: Vicenç Soler Ruíz i Rongrong Zhang
• Curs 2025/26

2 METODOLOGIA I PLANIFICACIÓ

2.1 Metodologia

La metodologia del projecte combina enfocaments Agile [3] i Lean Startup [4] amb l'objectiu de seguir un procés iteratiu i adaptatiu basat en l'aprenentatge continu. El treball s'organitza en cicles successius de recerca, implementació, avaluació i reajust dels objectius.

Aquest enfocament és especialment adequat per tractar tecnologies en evolució constant, ja que permet adaptar la direcció del projecte segons els resultats obtinguts. La metodologia Agile s'utilitza per estructurar els objectius i dividir el treball en iteracions curtes, mentre que Lean Startup facilita la validació progressiva de les solucions desenvolupades.

En cada iteració es construeix una versió mínima funcional, s'avalua el seu rendiment i s'utilitzen els resultats per millorar i redefinir la següent fase del projecte.

2.2 Planificació

Com s'ha esmentat a l'apartat de metodologia, el projecte s'ha organitzat en sprints iteratius, seguint un enfocament inspirat en Agile i Lean Startup. Cada sprint correspon a una fase d'experimentació, validació i millora progressiva del sistema.

A continuació es detalla la planificació seguida durant el desenvolupament del projecte:

Sprint 1 – Definició del problema i experiment inicial

- Objectius del projecte i metodologia de treball
- Recerca d'informació sobre LLMs i Text-to-SQL
- Preparació de l'entorn de treball i BBDD inicials
- Implementació del primer model de generació de SQL
- Disseny i execució de l'experiment inicial basat en prompt engineering
- Anàlisi de resultats inicials i detecció de limitacions

Sprint 2 – Disseny d'una bateria de proves i introducció al RAG

- Definició de mètriques d'avaluació
- Sistema de benchmarking de consultes
- Implementació de la pipeline utilitzant RAG
- Execució de proves sobre els primers models
- Anàlisis dels resultats
- Elaboració de conclusions del segon experiment

Sprint 3 – Millora de la bateria de proves i paral·lelització

- Actualització RAG i paral·lelització de les execucions
- Noves mètriques de avaluació
- Ampliació de benchmarking de consultes
- Comparació de rendiment entre models
- Anàlisis dels resultats
- Elaboració de conclusions del tercer experiment.

Sprint 4 – Introducció de LLM com a jutge i una base de dades real

- Disseny de l'extensió de la pipeline
- Definició de la nova mètrica d'avaluació
- Execució de la bateria de proves amb diferents models
- Execució amb una base de dades real i més gran
- Anàlisi de resultats i conclusions

Sprint 5 – Sistema final i aplicació offline

- Disseny del sistema final de xat offline Text-to-SQL
- Integració del millor model i pipeline complet
- Optimització del sistema per execució local
- Proves finals d'ús i validació funcional
- Redacció de conclusions finals del projecte

3 ESTAT DE L'ART

3.1 Models de llenguatge de gran escala

Els *Large Language Models* (LLM) són models d'intel·ligència artificial dissenyats per comprendre i generar llenguatge natural. La majoria dels models actuals es basen en l'arquitectura *Transformer*, presentada per Vaswani et al. el 2017, que va suposar un canvi de paradigma en el camp del processament del llenguatge natural.

Aquests models s'entrenen amb grans quantitats de dades textuals mitjançant tècniques d'aprenentatge autosupervisat. Durant aquest procés aprenen patrons estadístics i relacions entre paraules, frases i conceptes, fet que els permet predir el text més probable a continuació d'una seqüència determinada. Com a resultat, són capaços de realitzar tasques molt diverses, com ara respondre preguntes, resumir documents, traduir textos o generar codi font.

L'augment de la mida dels models, tant pel que fa al nombre de paràmetres com a la quantitat de dades utilitzades durant l'entrenament, ha comportat una millora significativa en la qualitat de les respostes. Tanmateix, aquests models també presenten limitacions, com ara la generació d'informació incorrecta (*hallucinations*), la dependència del context proporcionat i l'elevat cost computacional necessari tant per entrenar-los com per executar-los.

3.2 SQL

SQL (*Structured Query Language*) és el llenguatge estàndard utilitzat per interactuar amb bases de dades relacionals. Va ser desenvolupat durant la dècada dels setanta a partir del model relacional proposat per Edgar F. Codd i, actualment, és suportat per la majoria dels sistemes de gestió de bases de dades.

Aquest llenguatge permet realitzar operacions de consulta, inserció, modificació i eliminació de dades, així com gestionar l'estructura de les bases de dades i controlar els permisos d'accés. Les consultes SQL es construeixen combinant diferents instruccions, entre les quals destaquen `SELECT`, `FROM`, `WHERE`, `JOIN`, `GROUP BY` i `ORDER BY`.

Malgrat la seva sintaxi relativament senzilla, la construcció de consultes complexes requereix conèixer l'estructura de la base de dades, les relacions entre les diferents taules i el funcionament dels operadors disponibles.

3.3 Retrieval-Augmented Generation

El *Retrieval-Augmented Generation* (RAG) [5] és una tècnica que combina models generatius amb sistemes de recuperació d'informació. El seu objectiu és complementar el coneixement del model amb informació externa abans de generar una resposta.

El funcionament d'un sistema RAG es divideix habitualment en dues etapes. En primer lloc, un sistema de recuperació identifica els documents més rellevants dins d'una col·lecció d'informació utilitzant tècniques de cerca semàntica basades en *embeddings*. Posteriorment, aquests documents s'incorporen al context del model generatiu, que els utilitza per elaborar una resposta més precisa i fonamentada.

Aquest enfocament presenta diversos avantatges respecte als models generatius tradicionals. Permet utilitzar informació actualitzada sense necessitat de reentrenar el model, redueix la probabilitat de generar respostes incorrectes i facilita l'adaptació dels models a dominis específics mitjançant la incorporació de documentació especialitzada.

No obstant això, el rendiment dels sistemes RAG depèn directament de la qualitat del procés de recuperació. Si els documents recuperats no són rellevants o són incomplets, la qualitat de la resposta generada també es veu afectada.

3.4 Embeddings i cerca semàntica

Els *embeddings* són representacions vectorials de paraules, frases o documents en un espai de dimensions elevades. Aquestes representacions permeten capturar relacions semàntiques entre textos, de manera que dos fragments amb significats similars tendeixen a situar-se propers dins de l'espai vectorial.

Els sistemes RAG utilitzen habitualment aquests vectors per implementar mecanismes de cerca semàntica. Quan un usuari formula una consulta, aquesta es transforma en un embedding i es compara amb els embeddings dels documents disponibles mitjançant mesures de similitud, com ara la similitud cosinus o la distància euclidiana. D'aquesta manera és possible recuperar informació relacionada pel seu significat, encara que no comparteixi exactament les mateixes paraules.

Aquest tipus de cerca representa una evolució respecte a la cerca tradicional basada únicament en coincidències lèxiques i constitueix un dels elements fonamentals dels sistemes RAG actuals.

4 PUNT DE PARTIDA

Abans d'iniciar els experiments, és necessari definir l'entorn de treball i els recursos disponibles durant el desenvolupament del projecte. Per a la realització dels experiments, es disposa d'un clúster equipat amb tres GPU (dues NVIDIA GeForce RTX 2080 i una 3090) que permet executar localment diferents models de llenguatge de gran escala prèviament descarregats. Aquest entorn facilita la comparació del rendiment de diversos models sense dependre de serveis externs.

Pel que fa a les dades, es compta inicialment amb una base de dades relacional formada per quatre taules. Tot i les seves dimensions reduïdes, presenta una estructura suficientment representativa per generar consultes de diferent complexitat i avaluar la capacitat dels models per interpretar l'esquema i generar consultes SQL correctes.

4.1 Models disponibles

Com s'ha comentat anteriorment, el clúster de càlcul disposa de diversos models de llenguatge de codi obert ja descarregats i preparats per ser utilitzats en els experiments del projecte. Aquests models permeten avaluar diferents capacitats de generació de codi i consultar bases de dades en entorns controlats, sense dependència de serveis externs.

Un dels factors més rellevants en aquests models és el nombre de paràmetres, ja que està directament relacionat amb la seva capacitat d'aprenentatge i expressivitat. En general, models amb un nombre més elevat de paràmetres tendeixen a obtenir millors resultats en tasques complexes, com la generació de consultes SQL, tot i que també requereixen més recursos computacionals i temps d'execució. Per contra, models més petits són més eficients, però poden mostrar limitacions en la comprensió de contextos complexos o esquemes de bases de dades extensos.

Els models utilitzats en aquest entorn s'executen mitjançant *Ollama*, una plataforma que permet descarregar, gestionar i executar models de llenguatge de forma local. Aquesta eina simplifica la seva integració en entorns de desenvolupament, facilitant l'experimentació amb diferents arquitectures i versions de models sense necessitat d'utilitzar APIs externes. A més, l'execució local garanteix un major control sobre els recursos utilitzats i evita dependències de tercers, fet especialment rellevant en entorns d'investigació i benchmarking.

A continuació podem veure els models seleccionats per els experiments.

Model	Paràmetres	Espai necessari
llama3.2	3B	2.0 GB
llama3.1	8B	4.9 GB
llama3.3	70B	42 GB
deepseek-coder-v2	16B	8.9 GB
qwen2.5-coder	7B	4.7 GB
qwen2.5-coder	32B	19 GB

4.2 Base de dades inicial

La base de dades utilitzada en aquest treball ha estat proporcionada pel tutor amb l'objectiu de servir com a entorn controlat per a la realització de les proves experimentals. Es tracta d'una base de dades amb dades o reals, de mida reduïda i estructura relativament senzilla, però dissenyada de manera que incorpora suficient complexitat per permetre l'avaluació progressiva de les capacitats del model. Concretament, la seva estructura relacional inclou diverses taules interconnectades mitjançant claus primàries i foranes, així com diferents tipus de dades, fet que permet formular consultes de dificultat creixent. Aquesta simplicitat controlada resulta especialment adequada per al plantejament del treball, ja que facilita la definició d'un conjunt de preguntes incrementals que permeten posar a prova el model de manera gradual, des de consultes bàsiques fins a operacions més complexes com joins, filtratges condicionals o agregacions.

A continuació es mostra l'esquema de la base de dades utilitzada en els experiments, on es poden observar les taules principals i les seves relacions:

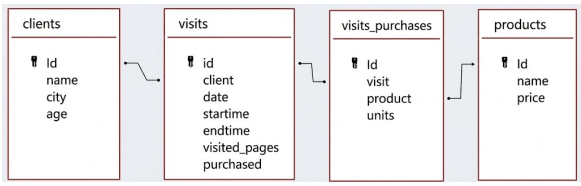

Fig. 1: Estructura de la base de dades

El conjunt de taules de la base de dades forma un model relacional que connecta clients, visites i compres. A la taula **clients** s'hi recullen les dades bàsiques de cada persona (identificador, nom, ciutat i edat). La taula **visits** registra cada visita d'un client, incloent-hi la data, l'hora d'inici i final, les pàgines visitades i si s'hi ha efectuat alguna compra. Quan una visita genera una compra, aquesta queda detallada a **visits purchases**, on s'indica quin producte s'ha adquirit i en quina quantitat. Finalment, la taula **products** conté el catàleg d'articles disponibles, amb el seu identificador, nom i preu. En conjunt, l'esquema permet seguir el recorregut complet d'un client: des de la seva informació personal fins al seu comportament de navegació i les compres associades.

5 PRIMER EXPERIMENT

L'objectiu del primer experiment és analitzar les capacitats inicials d'un model de llenguatge en la generació de consultes SQL, utilitzant tècniques de *prompt engineering* [6]. En concret, es pretén estudiar fins a quin punt la qualitat del prompt pot influir en els resultats obtinguts i determinar les limitacions del model quan no es disposa de context addicional ni mecanismes de suport externs. Aquest experiment permet establir una primera línia base de rendiment i avaluar si l'ús exclusiu del prompting és suficient per abordar el problema Text-to-SQL o si és necessari incorporar estratègies més avançades. Per a aquest primer experiment s'ha seleccionat el model *LLaMA 3.2* de 3B paràmetres. Aquesta elecció es basa principalment en el fet que es tracta d'un model relativament lleuger, que permet una execució eficient en entorns amb recursos limitats, facilitant així una alta velocitat d'inferència i un menor consum de memòria GPU. Cal destacar també que es tracta d'un model de propòsit general, no específicament entrenat per a la generació de codi o consultes SQL, fet que permet avaluar fins a quin punt un model generalista és capaç de resoldre el problema Text-to-SQL sense una especialització prèvia en aquest domini.

5.1 Conjunt de proves

Per a la realització de l'avaluació s'ha definit un conjunt de proves reutilitzable, que s'ha anat ampliant al llarg del projecte segons les necessitats dels diferents experiments. Aquest conjunt està estructurat en diversos nivells de complexitat amb l'objectiu d'analitzar de manera progressiva la capacitat del model per generar consultes SQL a partir de llenguatge natural. Aquesta estratificació permet observar l'evolució del rendiment des de casos senzills fins a escenaris que requereixen un major grau de raonament i comprensió de l'estructura de la base de dades.

El nivell 1 inclou consultes bàsiques sobre una única taula, centrades en operacions de selecció i filtratge simple, amb l'objectiu d'avaluar la generació correcta de sintaxi SQL elemental.

El nivell 2 introdueix condicions més específiques i subconsultes senzilles, incorporant una primera aproximació a relacions implícites entre dades.

El nivell 3 incorpora operacions de *JOIN*, exigint al model la capacitat de relacionar taules mitjançant claus i interpretar estructures relacionals. En aquest nivell, en alguns casos es prova també l'impacte de modificacions en el *prompt*.

El nivell 4 augmenta la complexitat mitjançant múltiples relacions entre taules i combinacions de filtres, requerint una comprensió més global de l'esquema de dades.

El nivell 5 introdueix operacions d'agregació, com comptatges i mitjanes, juntament amb l'ús de *GROUP BY*, orientades a consultes de tipus analític.

El nivell 6 amplia aquestes agregacions amb càlculs derivats entre taules i operacions de sumatori, incrementant la necessitat de consistència entre entitats.

Finalment, el nivell 7 recull les consultes més complexes, incloent subconsultes avançades, condicions d'exclusió i selecció de resultats òptims, amb l'objectiu de posar a prova el raonament SQL més complet del model.

5.2 Codi del experiment

En aquest primer experiment, la implementació del codi és senzilla i es basa principalment en l'ús d'una funció encarregada d'executar el model, tal com es pot observar a la Figura A.1. La metodologia seguida consisteix a modificar manualment cada pregunta i comprovar que la consulta SQL generada pel model s'executa correctament en SQLite, retornant el resultat esperat. Cal destacar que aquesta metodologia és lenta i poc eficient, ja que requereix l'actualització manual del *prompt* i la verificació individual de cada cas. No obstant això, en absència d'un sistema d'avaluació automàtic en aquesta fase inicial, constitueix l'opció més adequada per analitzar de manera controlada el comportament del model. Per començar a fer les proves només ens falta un *prompt* inicial.

5.3 Resultats de l'experiment inicial

En aquest apartat es presenten els resultats obtinguts durant el primer experiment. Com a referència inicial, el *prompt* utilitzat es mostra a la Figura A.2. A continuació es detallen els resultats organitzats per nivells de complexitat.

Nivell 1 — Consultes bàsiques sobre una sola taula

Les primeres proves consistien en seleccions i filtratges simples sobre una única taula. El model va respondre correctament en tots els casos, tant sintàcticament com semànticament.

Els resultats van validar que el model comprenia l'estructura bàsica de l'esquema i era capaç de recuperar informació directa sense operacions relacionals. L'única diferència

observada fou l'absència del punt i coma final, sense cap implicació funcional.

Exemple:
Pregunta: List the ages of all clients
Solució model:

```
SELECT age FROM clients
```

Nivell 2 — Filtratge semàntic i interpretació parcial

En aquest nivell es van introduir condicions que requerien una interpretació més acurada de la pregunta. Van aparèixer les primeres desviacions.

En el primer cas, el model va retornar columnes addicionals no sol·licitades (city, age), tot i que la condició de filtratge era correcta. En el segon cas, va optar per una subconsulta sobre `visits_purchases` en lloc d'emprar directament la columna booleana `purchased`. La resposta era funcionalment equivalent però estructuralment diferent de l'esperada, i evidenciava una capacitat de raonament relacional incipient. Aquests errors van motivar la necessitat de restringir la selecció d'atributs de forma més explícita.

Exemple:
Pregunta: List the names of clients younger than 25
Solució model:

```
SELECT name, city, age FROM clients
WHERE age < 25;
```

Nivell 3 — Primeres unions entre taules

Aquest nivell introduïa operacions `JOIN` simples entre dues taules.

El model va identificar correctament les claus foranes i va generar una unió sintàcticament vàlida, emprant àlies genèrics (T1, T2) en lloc dels usuals per convenció. El resultat era funcionalment equivalent a la solució de referència.

Exemple:
Pregunta: List client names and the number of pages they visited
Solució model:

```
SELECT T1.name, T2.visited_pages
FROM clients AS T1
INNER JOIN visits AS T2 ON
T1.id = T2.client;
```

Refinament del prompt (primera iteració)

A partir dels errors detectats als nivell 4, es va reformular el prompt per reforçar el procés de *schema linking*. Les modificacions principals van ser:

- Descripcions semàntiques *inline* a cada columna (p. ex. `purchased BOOLEAN -- true if at least one product was purchased`).
- Restriccions de pertinença: indicació explícita de quines columnes pertanyen a cada taula (p. ex. `visits_purchases` does not contain product names).
- Instruccions per evitar duplicats: *When returning entity names, avoid duplicates unless explicitly requested*.
- Instruccions per limitar la selecció: *Return only the columns explicitly requested in the question* i *Do not include additional columns*.

Addicionalment, es va comprovar que la reformulació de les preguntes tenia un impacte notable sobre la qualitat de les respostes. Expressions més específiques i explícites van produir resultats considerablement més precisos que formulacions ambigües.

Nivell 4 — Consultes multi-taula i deduplicació

La incorporació de `DISTINCT` i la limitació de columnes van corregir els problemes de duplicació i d'informació no sol·licitada. La reformulació de la segona pregunta —de "List products bought by each client" a "List the names of products purchased by each client"— va eliminar la tendència del model a generar agregacions no demanades.

Exemple:
Pregunta: List the names of products that were purchased at least once
Solució model:

```
SELECT p.name
FROM products p
JOIN visits_purchases vp ON
p.id = vp.product;
```

Nivell 5 — Agregacions simples

Les agregacions globals van ser resoltes perfectament. En canvi, la consulta agrupada per entitat va ometre inicialment el nom del client en el resultat. La reformulació de la pregunta per incloure explícitament "List each client name and..." va corregir completament el comportament.

Exemple:
Pregunta: Show the maximum product price
Solució model:

```
SELECT MAX(price) FROM products;
```

Nivell 6 — Agregacions relacionals complexes

En aquest nivell calia combinar múltiples unions amb operacions d'agregació sobre diverses taules.

L'error més rellevant va ser l'associació incorrecta de la columna `units` amb la taula `visits` en lloc de `visits_purchases`. Per corregir-ho es va afegir al prompt la indicació explícita "Quantity purchased is stored only in visits_purchases.units" i un exemple de *few-shot prompting* amb la consulta de diners generats per producte. Els resultats van millorar parcialment, tot i que la segona consulta encara presentava una vinculació incorrecta entre clients i `visits_purchases`, saltant-se la taula `visits` en la cadena de relacions.

Exemple:
Pregunta: For each client, show their name and total number of purchased units
Solució model:

```
SELECT c.name, SUM(v.units)
FROM clients c
JOIN visits v ON c.id = v.client
GROUP BY c.name;
```

Nivell 7 — Consultes avançades

El darrer nivell incloïa subconsultes, condicions negatives i criteris d'ordenació.

El comportament va ser heterogeni. En la consulta de clients sense compres, el model va generar un `LEFT JOIN` amb filtratge per `NULL`, que és un patró vàlid però que no detecta el cas de clients amb visites però sense compres associades, produint un resultat parcialment incorrecte. En la consulta del producte més car, va ordenar per `units * price` en lloc de per `price` en solitari, alterant la semàntica de la pregunta. En canvi, la consulta del client que més va gastar va ser resolta correctament, tot i incloure una columna addicional a la selecció.

Exemple:

Pregunta: List the names of clients who visited more than once

Solució model:

```
SELECT c.name, COUNT(v.id)
FROM clients c
JOIN visits v ON c.id = v.client
GROUP BY c.name
HAVING COUNT(v.id) > 1;
```

5.4 Conclusió

El model va demostrar un rendiment sòlid en consultes de selecció directa, filtratge simple i agregacions globals, resolent-les de manera consistent i sense desviacions significatives. La incorporació progressiva de descripcions semàntiques, restriccions de pertinença i instruccions explícites, recollides al prompt final representat a la figura **??**, va comportar una millora notable en la precisió de les respostes, especialment en tasques que requerien identificar correctament l'origen de cada atribut dins de l'esquema.

Tanmateix, les limitacions més marcades van aparèixer en consultes que exigien una vinculació fina entre columna i taula, així com en la interpretació de condicions lògiques més abstractes, com ara la negació, l'absència d'esdeveniments o la selecció d'elements extrems. Aquestes dificultats indiquen que, malgrat que el model és capaç de reproduir patrons SQL complexos, encara presenta mancances en situacions que requereixen inferència lògica més elaborada i un raonament relacional precís.

A partir d'aquest punt, continuar ampliant o sofisticant el prompt no sembla una via rendible: la complexitat necessària per compensar les limitacions de raonament del model esdevé massa elevada i poc consistent, i no garanteix una millora proporcional en el comportament. Per tant, els resultats suggereixen que el límit pràctic del refinament del prompt ja s'ha assolit, i que superar aquestes mancances requeriria estratègies més enllà de l'enginyeria de prompts.

6 SEGON EXPERIMENT

L'objectiu d'aquest segon experiment és automatitzar el procés d'avaluació desenvolupat durant el primer experiment i ampliar-lo per poder comparar simultàniament diversos models de llenguatge. Per aconseguir-ho es desenvolupa una pipeline que, de manera automàtica, genera la consulta SQL a partir d'una pregunta en llenguatge natural, executa la consulta sobre la base de dades i registra diferents mètriques de rendiment per a cada model.

A diferència del primer experiment, en què el *prompt* es construïa manualment, en aquesta fase s'utilitza la cadena `create_sql_query_chain` proporcionada per LangChain [7]. Aquesta cadena recupera automàticament l'esquema de la base de dades a partir de l'objecte `SQLDatabase` i l'incorpora al *prompt* que s'envia al model. Tot i que aquest mecanisme no correspon a una implementació completa de *Retrieval-Augmented Generation* (RAG), comparteix la mateixa filosofia: proporcionar al model únicament el context necessari en el moment de la generació de la resposta, evitant haver d'incloure manualment tota la informació de l'esquema en cada consulta.

Aquest enfocament aporta diversos avantatges. En primer lloc, simplifica considerablement la implementació de la pipeline, ja que el context de la base de dades es genera de manera automàtica. A més, facilita la reutilització del sistema amb diferents bases de dades sense necessitat de modificar el *prompt*, millorant així la flexibilitat de la solució.

Finalment, aquesta pipeline permet executar la mateixa bateria de preguntes sobre diversos models de forma completament automàtica. Per a cada consulta es registra el temps de generació, el temps total d'execució, si la consulta SQL és executable i una primera mètrica basada en la presència de paraules clau esperades dins de la consulta generada. Aquesta automatització permet obtenir una primera comparativa objectiva entre models i constitueix la base sobre la qual es desenvoluparan les següents versions de la pipeline, incorporant progressivament noves estratègies de recuperació de context i mètriques d'avaluació més robustes.

6.1 Arquitectura de la pipeline

La pipeline implementada es basa en Python i integra la biblioteca `LangChain` amb models locals executats mitjançant `Ollama`. L'objectiu és generar consultes SQL a partir d'un conjunt de preguntes i executar-les sobre una base de dades SQLite per posteriorment avaluar-ne la qualitat.

6.1.1 Base de dades

La base de dades utilitzada és una base SQLite carregada des del fitxer `db_clients_FK.db`. Aquesta base de dades es connecta de dues maneres: d'una banda, mitjançant l'abstracció `SQLDatabase` de LangChain per facilitar la generació de consultes, i de l'altra, mitjançant una connexió directa amb `sqlite3` per a l'execució de les consultes generades.

6.1.2 Bateria de preguntes

La bateria de preguntes està definida a la variable `QUESTIONS`, que conté un conjunt d'instàncies amb identificador, text de la pregunta, dificultat i una llista de paraules clau esperades dins la consulta SQL. Aquestes paraules clau s'utilitzen posteriorment com a heurística per estimar la qualitat de la consulta generada.

6.1.3 Models de llenguatge

L'experiment compara diversos models de llenguatge executats localment mitjançant `Ollama`. Entre els models considerats hi ha variants de `Qwen2.5-Coder` i `DeepSeek-Coder` que son models especialitzats en la generació de codi. També variants del model `llama` son més generalistes sense estar especialitzats. Cada model s'inicialitza amb temperatura zero per reduir la variabilitat en les respostes i garantir la reproductibilitat dels resultats. L'objectiu és comparar la diferència de rendiment entre models especialitzats i generals, i comparar variants del mateix model on varia la quantitat de paràmetres.

6.1.4 Generació de consultes SQL

La generació de consultes SQL es realitza mitjançant la funció `create_sql_query_chain` de LangChain, que construeix una cadena de raonament entre el model de llenguatge i l'esquema de la base de dades.

Un cop el model genera la consulta, aquesta passa per una fase de neteja implementada a la funció `neteja_sql`. Aquesta funció elimina marques de format com ```` ```sql ````, etiquetes pròpies de LangChain (per exemple `SQLQuery:`, `Answer:`) i extreu únicament la sentència SQL rellevant mitjançant expressions regulars. També es trunca qualsevol contingut posterior al primer punt i coma.

6.1.5 Execució de les consultes

Les consultes SQL netejades s'executen directament sobre la base de dades mitjançant la funció `executa_sql`, que utilitza un cursor de `sqlite3`. Aquesta funció captura tant els resultats de la consulta com possibles errors d'execució, els quals s'utilitzen posteriorment per determinar la correcció de la resposta.

6.1.6 Mètriques d'avaluació

L'avaluació dels resultats es basa en diverses mètriques:

- **Correcció de la consulta**: determinada per l'absència d'errors en l'execució SQL.
- **Score de paraules clau**: mesura heurística basada en la presència de paraules clau esperades dins la consulta generada.
- **Temps de generació**: temps necessari perquè el model generi la consulta SQL.
- **Temps total**: temps total des de la pregunta fins a l'obtenció del resultat final.
- **Nombre de resultats**: files retornades per la consulta executada.

El score de paraules clau es calcula mitjançant la funció `calcula_score_sql`, que compara la consulta generada amb una llista de termes esperats per a cada pregunta.

6.1.7 Emmagatzematge dels resultats

Els resultats de l'experiment s'emmagatzemen incrementalment en una estructura de tipus llista de diccionaris. Cada entrada conté informació sobre el model, la pregunta, la consulta SQL generada, els temps d'execució i les mètriques associades. Finalment, aquesta informació es converteix en un `DataFrame` de `Pandas` i s'exporta a un fitxer CSV anomenat `benchmark_resultats.csv`.

6.2 Metodologia

La metodologia experimental consisteix a executar cada model sobre totes les preguntes definides a la bateria, repetint el mateix procés de manera sistemàtica. Per a cada iteració es generen consultes SQL, s'executen sobre la base de dades i s'avaluen segons les mètriques definides.

Per garantir la comparabilitat entre models, es fixa la temperatura dels LLM a zero i es manté constant el mateix entorn d'execució. A més, es registra el temps de generació i execució per tal d'analitzar no només la qualitat sinó també l'eficiència dels models.

6.3 Resultats

Podem veure els resultats del experiment en la figura A.6

7 TERCER EXPERIMENT

Aquest tercer experiment amplia la pipeline anterior amb un enfocament orientat a l'eficiència computacional i a l'aprofitament de recursos hardware. En concret, la principal diferència respecte els experiments previs és la paral·lelització de l'execució dels models, permetent utilitzar simultàniament tres GPUs diferents.

Aquesta decisió és clau per reduir el temps total d'avaluació i, al mateix temps, permetre l'execució de models de diferent mida sense saturar una única GPU. Tal com es mostra a la Figura A.4, cada model és assignat a una GPU específica en funció de la seva mida i requeriments de memòria.

GPU	Memòria
GPU 0	11 GB
GPU 1	11 GB
GPU 2	24 GB

TAULA 1: CONFIGURACIÓ DE LES GPUS UTILITZADES EN EL TERCER EXPERIMENT

7.1 Arquitectura de la pipeline

La pipeline implementada combina generació de SQL amb un sistema RAG (Retrieval-Augmented Generation) i execució paral·lela de models mitjançant múltiples processos. El flux general consisteix en la càrrega del context de base de dades, la recuperació d'informació rellevant i la generació de consultes SQL a partir de preguntes en llenguatge natural.

7.1.1 Base de dades

La base de dades utilitzada és la mateixa que en experiments anteriors, basada en SQLite i carregada des del fitxer `db_clients_FK.db`. Cada procés de model crea la seva pròpia connexió independent per evitar conflictes durant l'execució paral·lela.

7.1.2 Bateria de preguntes

La bateria de preguntes es defineix a la variable `QUESTIONS`, i conté instàncies amb identificador, dificultat, pregunta i una consulta SQL de referència (`gold_sql`). Aquesta darrera s'utilitza per avaluar la qualitat de les respostes del model mitjançant comparació directa amb el resultat esperat.

7.1.3 Generació de consultes SQL

La generació de consultes SQL es realitza mitjançant una cadena de LangChain que combina un prompt estructurat amb un sistema de recuperació de context (RAG). Aquest prompt força el model a respondre únicament amb la consulta SQL, utilitzant l'esquema de la base de dades com a context.

El sistema RAG es construeix carregant un fitxer de text amb l'esquema de la base de dades (`schema.txt`), que posteriorment és fragmentat en trossos mitjançant `RecursiveCharacterTextSplitter`. Aquests fragments es transformen en embeddings amb `sentence-transformers/all-MiniLM-L6-v2` i s'emmagatzemen en una base vectorial FAISS per permetre la recuperació semàntica.

7.1.4 Execució de les consultes

Les consultes generades s'executen sobre SQLite mitjançant una funció segura que retorna tant les columnes com les files del resultat, així com possibles errors. En paral·lel, també s'executa la consulta de referència (`gold_sql`) per poder comparar els resultats.

Aquesta execució independent permet calcular mètriques basades no només en la sintaxi de la consulta, sinó també en la seva semàntica.

7.1.5 Mètriques d'avaluació

En aquest experiment s'utilitzen dues mètriques principals:

- **Execution Accuracy**: mesura si el conjunt de resultats retornats per la consulta del model coincideix exactament amb els resultats de la consulta de referència.
- **Comparació de columnes**: avalua si les columnes retornades són compatibles amb les esperades i si els resultats projectats coincideixen.
- **Temps de generació**: temps necessari per generar la consulta SQL.
- **Temps total**: temps complet del procés des de la pregunta fins a la validació final.

Les funcions `compute_execution_accuracy` i `comparacio_columnes` implementen aquestes mètriques, tal com es mostra a la Figura 2.

Fig. 2: Implementació de les mètriques d'avaluació

7.1.6 Emmagatzematge dels resultats

Els resultats de cada model s'emmagatzemen en un `DataFrame` independent i posteriorment es combinen en un únic conjunt de dades. Finalment, es genera un fitxer CSV anomenat `benchmark_paralel.csv`, que conté totes les execucions, mètriques i errors detectats.

7.2 Metodologia

La metodologia es basa en l'execució paral·lela de fins a tres models simultàniament, cadascun assignat a una GPU diferent segons el diccionari `GPU_MAP`. Aquest disseny permet optimitzar l'ús dels recursos disponibles i reduir significativament el temps total d'execució de l'experiment.

Cada model s'executa en un procés independent mitjançant `ProcessPoolExecutor`, assegurant així l'aïllament de memòria i evitant conflictes entre execucions.

Per a cada pregunta del conjunt `QUESTIONS`, es genera una consulta SQL, s'executa sobre la base de dades i es compara amb la resposta de referència. Aquest procés es repeteix per a tots els models en paral·lel.

7.3 Resultats

Podem veure els resultats del experiment en la figura A.7

8 QUART EXPERIMENT

Després d'obtenir els resultats del tercer experiment, es planteja una nova fase d'avaluació amb l'objectiu d'incorporar una mètrica capaç d'analitzar la qualitat semàntica de les consultes SQL generades. Les mètriques utilitzades fins aquest moment, com ara *Execution Accuracy* i la comparació de columnes, permeten verificar si una consulta retorna el mateix resultat que la consulta de referència sobre una instància concreta de la base de dades. No obstant això, aquestes mètriques presenten una limitació important: dues consultes poden retornar el mateix resultat sobre les dades disponibles i, tanmateix, no ser semànticament equivalents per a qualsevol altra instància de la base de dades.

Per superar aquesta limitació, en aquest experiment s'incorpora el paradigma *LLM-as-a-Judge* [8]. Aquest enfocament consisteix a utilitzar un model de llenguatge no per generar consultes SQL, sinó per analitzar dues consultes existents i determinar si són semànticament equivalents tenint en compte l'esquema complet de la base de dades.

8.1 Arquitectura de la pipeline

La pipeline d'aquest experiment és considerablement més senzilla que les anteriors, ja que no es tornen a generar consultes SQL. En el seu lloc, es reutilitzen els resultats obtinguts durant el tercer experiment i s'avaluen mitjançant un model de llenguatge que actua com a jutge.

8.1.1 Resultats d'entrada

Com a punt de partida s'utilitza el fitxer `benchmark_paralel.csv`, generat durant el tercer experiment. Aquest fitxer conté, per a cada pregunta i model, la consulta SQL generada, la consulta SQL de referència i les diferents mètriques calculades fins aquell moment.

D'aquesta manera, aquesta fase no requereix tornar a executar els models generadors de SQL, sinó que únicament analitza les consultes ja obtingudes.

8.1.2 Model LLM com a Judge

Per a la tasca d'avaluació s'ha seleccionat el model `Qwen2.5-Coder:7B`, executat localment mitjançant Ollama amb temperatura igual a zero.

L'elecció d'aquest model respon principalment a tres motius:

- És un model especialitzat en tasques de programació i comprensió de codi, mostrant un bon rendiment en l'anàlisi de consultes SQL.
- El model disposa d'una mida moderada (7 mil milions de paràmetres), fet que permet obtenir inferències ràpides mantenint una bona qualitat en les respostes.
- L'execució local garanteix la reproductibilitat de l'experiment i evita la dependència de serveis externs o APIs comercials.

En aquest experiment el model no actua com a generador de consultes SQL, sinó exclusivament com a avaluador, analitzant si dues consultes diferents són equivalents des d'un punt de vista semàntic.

8.1.3 Prompt d'avaluació

El comportament del model està determinat pel prompt utilitzat durant la inferència, representat a la Figura **??**.

Fig. 3: Prompt utilitzat pel model LLM-as-a-Judge

El prompt proporciona al model tres elements principals:

- L'esquema complet de la base de dades.
- La consulta SQL de referència (`gold_sql`).
- La consulta SQL generada pel model (`candidate_sql`).

A partir d'aquesta informació, el model ha de determinar si ambdues consultes són semànticament equivalents, és a dir, si retornarien el mateix resultat per a qualsevol base de dades que compleixi l'esquema proporcionat.

Per facilitar el processament posterior, el model ha de respondre sempre amb un format estructurat format per una etiqueta (`LABEL`) i una justificació (`REASON`).

8.1.4 Procés d'inferència

Per a cada fila del fitxer de resultats, el sistema extreu la consulta SQL generada i la consulta SQL de referència.

Si el model original no ha generat cap consulta SQL, la resposta es classifica directament com a `NOT_EQUIVALENT`. En cas contrari, es construeix el prompt corresponent i es realitza una inferència amb el model seleccionat.

La resposta obtinguda és posteriorment processada per extreure únicament l'etiqueta i la justificació, validant que l'etiqueta pertanyi a algun dels valors admesos.

8.1.5 Mètrica de Semantic Equivalence

El model pot assignar una de les tres classificacions següents:

Classificació	Puntuació
EXACT_EQUIVALENT	1.0
PARTIAL_EQUIVALENT	0.5
NOT_EQUIVALENT	0.0

TAULA 2: CONVERSIÓ DE LES ETIQUETES SEMÀNTIQUES A PUNTUACIONS NUMÈRIQUES

Aquesta nova mètrica, anomenada *Semantic Equivalence*, proporciona una estimació del grau d'equivalència lògica entre les dues consultes SQL.

A diferència de l'*Execution Accuracy*, que únicament compara els resultats obtinguts sobre una instància concreta de la base de dades, aquesta mètrica intenta determinar si les dues consultes serien equivalents per a qualsevol base de dades compatible amb l'esquema proporcionat.

8.1.6 Emmagatzematge dels resultats

Un cop finalitzada l'avaluació, s'incorporen tres noves columnes al conjunt de dades:

- `semantic_equivalence`, que conté l'etiqueta assignada pel model.
- `semantic_score`, corresponent a la puntuació numèrica associada.
- `semantic_reason`, que emmagatzema la justificació proporcionada pel model.

Finalment, tots els resultats s'emmagatzemen al fitxer `benchmark_semantic.csv`, que amplia el conjunt de dades obtingut durant els experiments anteriors.

8.2 Metodologia

La metodologia seguida consisteix a reutilitzar totes les consultes SQL generades durant el tercer experiment. Per a cadascuna d'elles es construeix un prompt que incorpora l'esquema de la base de dades, la consulta SQL de referència i la consulta SQL generada pel model.

El model LLM seleccionat actua com a jutge independent i determina el grau d'equivalència entre ambdues consultes. Les etiquetes obtingudes es transformen posteriorment en una puntuació numèrica que permet comparar quantitativament el rendiment dels diferents models.

Aquest enfocament introdueix una capa d'avaluació complementària que no depèn exclusivament dels resultats obtinguts durant l'execució de les consultes, sinó que també té en compte el seu significat semàntic.

8.3 Resultats

Podem veure els resultats del l'experiment a la figura A.8

9 Conclusions

Al llarg dels experiments realitzats s'ha pogut observar que els models especialitzats en generació de codi, com les famílies **Qwen2.5-Coder** i **DeepSeek-Coder**, obtenen de manera consistent un millor rendiment en la tasca de Text-to-SQL que els models de propòsit general. Aquesta diferència es fa especialment evident en les consultes que requereixen una major comprensió de l'estructura SQL i de les relacions entre taules, on els models especialitzats mostren una major precisió i robustesa.

Pel que fa a la mida dels models, els resultats indiquen que les versions amb un nombre més elevat de paràmetres no proporcionen una millora significativa en la qualitat de les consultes generades per al nivell de complexitat de les preguntes utilitzades en aquest treball. En canvi, sí que impliquen un increment considerable del temps d'inferència i del consum de recursos computacionals. Això suggereix que, per a aquest conjunt d'experiments, existeix un bon compromís entre precisió i eficiència en models de mida intermèdia especialitzats en codi.

Com a línies de treball futures, seria interessant ampliar el conjunt de preguntes incorporant consultes de major complexitat, amb més subconsultes, funcions analítiques i operacions sobre múltiples taules. Igualment, seria convenient repetir els experiments sobre una base de dades més extensa i representativa d'un entorn real, fet que permetria avaluar la capacitat dels models en esquemes més complexos i amb un major volum d'informació.

Finalment, tot i que no ha estat possible completar els darrers objectius definits en la planificació inicial, com ara el desenvolupament del sistema final de consulta offline, es considera que els resultats obtinguts són satisfactoris. Durant el projecte s'han desenvolupat eines de benchmarking, pipelines d'avaluació i noves mètriques, com l'avaluació semàntica mitjançant un model *LLM-as-a-Judge*, que constitueixen una base sòlida per continuar la investigació. En aquest sentit, es considera que el projecte ha assolit els seus objectius principals i que estableix una metodologia robusta que permet seguir avançant en l'estudi dels models de llenguatge aplicats al problema Text-to-SQL.

Referències

1. Cloudflare. (2025). *What is a large language model (LLM)?* Cloudflare Learning Center. Available at: https://www.cloudflare.com/learning/ai/what-is-large-language-model/

2. Hong, Z., Yuan, Z., Zhang, Q., Chen, H., Dong, J., Huang, F., & Huang, X. (2024). *Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL*. arXiv preprint arXiv:2406.08426. Available at: https://arxiv.org/abs/2406.08426

3. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). *Manifesto for Agile Software Development*. Available at: https://agilemanifesto.org/

4. Ries, E. (2011). *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business. Available at: https://theleanstartup.com/book

5. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Advances in Neural Information Processing Systems (NeurIPS 2020), 33, 9459–9474.

6. OpenAI. (2024). *Prompt engineering*. OpenAI Documentation. Available at: https://platform.openai.com/docs/guides/prompt-engineering

7. LangChain. (2025). *LangChain Documentation*. Available at: https://python.langchain.com/

8. Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., & Stoica, I. (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena*. Advances in Neural Information Processing Systems (NeurIPS 2023). Available at: https://arxiv.org/abs/2306.05685

A Apèndix

```python
resp=ollama.generate(
    model="llama3.2:3b",
    prompt=prompt,
    options= {"temperature":0},
    keep_alive="0"
)

print(resp['response'])
```

Fig. A.1: Resultat de l'experiment

```
Context prompt:

Database schema:

Table clients(
    id INTEGER PRIMARY KEY,
    name TEXT,
    city TEXT,
    age INTEGER
)

Table visits(
    id INTEGER PRIMARY KEY,
    client INTEGER,
    date DATE,
    starttime TIME,
    endtime TIME,
    visited_pages INTEGER,
    purchased BOOLEAN
)

Table products(
    id INTEGER PRIMARY KEY,
    name TEXT,
    price FLOAT
)

Table visits_purchases(
    id INTEGER PRIMARY KEY,
    visit INTEGER,
    product INTEGER,
    units INTEGER
)

Relationships:
- visits.client = clients.id
- visits_purchases.visit = visits.id
- visits_purchases.product = products.id

Instructions:
- Use only the tables and columns provided
- Do not invent columns or tables
- Use correct SQL syntax
- Return only the SQL query

Question: List all clients
```

Fig. A.2: Prompt inicial

```
prompt = """Database schema:

Table clients(
    id INTEGER PRIMARY KEY,
    name TEXT, -- client name
    city TEXT, -- client city
    age INTEGER -- client age
)

Table visits(
    id INTEGER PRIMARY KEY,
    client INTEGER, -- references clients.id
    date DATE,
    starttime TIME,
    endtime TIME,
    visited_pages INTEGER,
    purchased BOOLEAN -- true if at least one product was purchased
)

Table products(
    id INTEGER PRIMARY KEY,
    name TEXT, -- product name
    price FLOAT -- product price
)

Table visits_purchases(
    id INTEGER PRIMARY KEY,
    visit INTEGER, -- references visits.id
    product INTEGER, -- references products.id
    units INTEGER -- quantity purchased
)

Relationships:
- visits.client = clients.id
- visits_purchases.visit = visits.id
- visits_purchases.product = products.id

Important:
- Product names are stored in products.name
- Client names are stored in clients.name
- visits_purchases does not contain product names
- Use joins when information is distributed across tables
- Quantity purchased is stored only in visits_purchases.units
- visits does not contain purchase quantity information

Instructions:
- Use only the tables and columns provided
- Do not invent tables or columns
- Verify that every column exists in the correct table
- Use valid SQL syntax
- Return ONLY the SQL query
- Do not include explanations
- When returning entity names, avoid duplicates unless explicitly requested
- Return only the columns explicitly requested in the question
- Do not include additional columns


Examples:

Question: For each product, show total money generated
SQL:
SELECT p.name, SUM(p.price * vp.units)
FROM products p
JOIN visits_purchases vp ON p.id = vp.product
GROUP BY p.name;

Now answer:

Question:  Client who spent the most money


"""
```

Fig. A.3: Prompt final

```
GPU_MAP = {
    "llama3.2:3b": "0",
    "llama3.1:8b": "1",
    "deepseek-coder-v2:16b": "1",
    "qwen2.5-coder:7b": "0",
    "qwen2.5-coder:32b": "2",
    "llama3.3:70b": "2"
}
```

Fig. A.4: Model per GPU

```
template = """
You are an expert SQL reviewer.

Database schema:

{schema}

Gold SQL query:

{gold_sql}

Candidate SQL query:

{candidate_sql}

Determine whether the two SQL queries are semantically equivalent.

Two queries are semantically equivalent if they always return the same result
for every valid database instance satisfying the schema.

Return your answer in this format:

LABEL: <EXACT_EQUIVALENT | PARTIAL_EQUIVALENT | NOT_EQUIVALENT>
REASON: <brief explanation>

Rules:
- If LABEL = EXACT_EQUIVALENT → REASON must be NONE
- If LABEL = PARTIAL_EQUIVALENT → explain what is correct and what is missing or different
- If LABEL = NOT_EQUIVALENT → explain why they are incorrect
- REASON must always be informative except for EXACT_EQUIVALENT
- Do not add any extra text outside this format
"""
```

Fig. A.5: Prompt del jutge

```
============================================================
RESUM PER MODEL
============================================================
                        correcte  temps_generacio  temps_total  score_keywords
model
deepseek-coder-v2:16b  0.979592         0.813388     0.814143        0.969388
llama3.1:8b            0.938776         0.658224     0.658898        0.928571
llama3.2:3b            0.795918         0.539959     0.540592        0.887755
llama3.3:70b           0.959184        32.999163    32.999551        0.942245
qwen2.5-coder:32b      1.000000         2.933367     2.933837        0.921837
qwen2.5-coder:7b       0.959184         0.683429     0.683878        0.942245
```

Fig. A.6: Resultats del experiment 2

```
===== RESUM =====
                       temps_generacio  temps_total  execution_accuracy  comparacio_columnes
model
deepseek-coder-v2:16b         5.425611     5.427861            0.638889             0.333333
llama3.1:8b                   4.527028     4.529250            0.638889             0.527778
llama3.2:3b                   4.539778     4.541556            0.555556             0.666667
llama3.3:70b                 36.584778    36.586472            0.666667             0.527778
qwen2.5-coder:32b            27.843278    27.845472            0.666667             0.583333
qwen2.5-coder:7b             10.534806    10.536917            0.694444             0.472222
```

Fig. A.7: Resultats del experiment 3

```
===== RESUM =====
                       execution_accuracy  semantic_score
model
deepseek-coder-v2:16b            0.638889        0.958333
llama3.1:8b                      0.638889        0.847222
llama3.2:3b                      0.555556        0.722222
llama3.3:70b                     0.666667        0.958333
qwen2.5-coder:32b                0.666667        0.958333
qwen2.5-coder:7b                 0.694444        0.958333
```

Fig. A.8: Resultats del experiment 4