الجمـهورية الجزائرية الديمقراطية الشعبيـة
وزارة التعليم العالي والبحث العلمي
جـامعـة سعـيـدة د. مـولاي الطـاهـر
كلية الرياضيات و الإعلام الآلي و الاتصالات السلكية و اللاسلكية
قسم: الإعلام الآلي

Mémoire de Master en informatique

Spécialité : Réseaux Informatiques et Systèmes Répartis (RISR)

T h è m e

ASK-Data : Conception et Évaluation d'un Assistant Local Text-to-SQL Basé sur les Small Language Models

▪ **Présenté par :**

MAHSAR Sara

TAHRI Imane

▪ **Dirigé par :**

Dr. CHAIBI Hassene

Année universitaire 2025-2026

Dédicace

♡ Dédicace ♡

Sara dédie ce travail à...

À mes chers parents, dont l'amour inconditionnel, leurs sacrifices et le soutien indéfectible ont été ma plus grande force tout au long de ce parcours.

À mes sœur et mon frère, complices de chaque instant, qui ont illuminé mes journées les plus difficiles.

À Joud et Nour, petits rayons de soleil, qui remplissent ma vie de joie, d'innocence et de bonheur.

À mes amis fidèles, pour leur présence, leurs encouragements et les rires partagés qui ont rendu ce chemin plus léger.

À tous ceux qui ont cru en moi, même quand je doutais de moi-même — cette réussite vous appartient aussi.

Imane dédie ce travail à...

À mes parents, piliers de ma vie, qui ont tout sacrifié pour que je puisse atteindre mes rêves — ce diplôme est autant le leur que le mien.

À ma famille, havre de paix et de chaleur, qui m'a toujours rappelé l'essentiel.

À mes amis sincères, témoins de mes doutes comme de mes victoires, merci d'avoir été là à chaque tournant.

À toutes les personnes qui, de près ou de loin, ont contribué à faire de cette aventure une réussite.

« L'effort d'aujourd'hui construit le succès de demain. »

Remerciements

♡ Remerciements ♡

À notre encadreur

Nous adressons nos plus sincères remerciements à **Dr. CHAIBI Hassene**, pour sa confiance, sa disponibilité permanente, ses conseils avisés et la rigueur de son encadrement. Sa bienveillance et son expertise ont été une source d'inspiration et de motivation tout au long de ce travail.

Au jury

Nous exprimons notre reconnaissance aux membres du jury pour l'honneur qu'ils nous font en acceptant d'évaluer ce travail et pour le temps précieux qu'ils lui ont consacré.

Au corps enseignant

Nous tenons à remercier l'ensemble du corps enseignant du Département Informatique de l'Université Dr. Moulay Tahar — Saïda pour la qualité de la formation dispensée, les connaissances transmises et les valeurs partagées tout au long de notre cursus.

À nos familles et amis

Un immense merci à nos familles pour leur patience, leur soutien moral et leur amour sans faille. À nos amis, pour leur présence, leurs encouragements et la belle énergie qu'ils nous ont apportée chaque jour.

« Merci du fond du cœur. »

Résumé

Ce mémoire s'inscrit dans le domaine de l'intelligence artificielle appliquée à l'interrogation des bases de données en langage naturel, et plus précisément dans la tâche de **Text-to-SQL**. L'objectif principal est de concevoir et d'évaluer un **assistant local basé sur des Small Language Models (SLM)** capable de traduire des requêtes en langage naturel en requêtes SQL exploitables, tout en garantissant des performances adaptées à des environnements à ressources limitées.

L'étude propose une évaluation comparative approfondie de plusieurs modèles de langage exécutés en local via LM Studio, incluant des modèles spécialisés SQL (SQLCoder-7B, SLM-SQL-1.5B, Gemma Text-to-SQL), des modèles généralistes (Qwen 3.5, DeepSeek-R1-Distill-Qwen) ainsi que des modèles Edge optimisés pour faible consommation (Ministral 3B, IBM Granite Tiny). Ces modèles couvrent différentes tailles, architectures et stratégies de quantification, permettant une analyse fine du compromis entre performance, précision et coût computationnel.

La méthodologie repose sur un protocole expérimental reproductible intégrant des scénarios de génération SQL à partir de requêtes multilingues, incluant des cas simples et complexes. Les performances sont évaluées selon plusieurs critères, notamment la précision syntaxique des requêtes générées, la validité sémantique, la capacité de généralisation ainsi que le temps d'inférence sur CPU.

Les résultats montrent que les modèles spécialisés SQL offrent une meilleure précision sur les requêtes structurées, tandis que certains modèles généralistes à plus grande capacité présentent de meilleures performances en raisonnement complexe. Les modèles Edge, bien que limités en précision, se distinguent par leur efficacité en termes de rapidité et de consommation de ressources, les rendant adaptés aux environnements embarqués.

Enfin, cette étude met en évidence l'importance du choix du modèle en fonction du contexte d'utilisation et souligne le potentiel des architectures hybrides et des techniques de distillation pour améliorer les systèmes Text-to-SQL locaux.

Mots clés : Text-to-SQL ; SLM (Small Language Models) ; Intelligence artificielle appliquée aux bases de données ; Assistants intelligents locaux.

الملخص

يندرج هذا البحث ضمن مجال الذكاء الاصطناعي المطبق على الاستعلام عن قواعد البيانات باستخدام اللغة الطبيعية، وبالتحديد في مهمة تحويل النص إلى SQL (Text-to-SQL). يتمثل الهدف الرئيسي في تصميم وتقييم مساعد محلي يعتمد على نماذج لغوية صغيرة (Small Language Models - SLMs) قادر على تحويل الاستعلامات المكتوبة باللغة الطبيعية إلى أوامر SQL قابلة للتنفيذ، مع ضمان أداء مناسب للبيئات ذات الموارد المحدودة.

تقدم هذه الدراسة تقييماً مقارناً معمقاً لعدة نماذج لغوية تعمل محلياً عبر منصة LM Studio، بما في ذلك نماذج متخصصة في SQL مثل SQLCoder-7B وSLM-SQL-1.5B ونماذج عامة مثل Qwen 3.5 وDeepSeek-R1-Distill-Qwen، بالإضافة إلى نماذج خفيفة موجهة للحافة (Edge) مثل Ministral 3B وIBM Granite Tiny. وتغطي هذه النماذج أحجاماً وبنى معمارية واستراتيجيات كمّ مختلفة، مما يسمح بتحليل دقيق للمفاضلة بين الأداء والدقة والتكلفة الحسابية.

تعتمد المنهجية على بروتوكول تجريبي قابل لإعادة الإنتاج يتضمن سيناريوهات توليد SQL من استعلامات متعددة اللغات، تشمل حالات بسيطة ومعقدة. ويتم تقييم الأداء وفق عدة معايير، من بينها الدقة التركيبية للاستعلامات الناتجة، والصحة الدلالية، والقدرة على التعميم، إضافة إلى زمن الاستجابة على وحدة المعالجة المركزية.

تُظهر النتائج أن النماذج المتخصصة في SQL تحقق دقة أعلى في الاستعلامات المهيكلة، بينما تقدم بعض النماذج العامة ذات القدرة الأكبر أداءً أفضل في مهام الاستدلال المعقدة. أما نماذج الحافة، ورغم محدودية دقتها، فتتميز بكفاءتها من حيث السرعة واستهلاك الموارد، مما يجعلها مناسبة للبيئات المدمجة.

وأخيراً، تؤكد هذه الدراسة أهمية اختيار النموذج وفقاً لسياق الاستخدام، وتبرز إمكانات البنى الهجينة وتقنيات التقطير لتحسين أنظمة Text-to-SQL المحلية.

الكلمات المفتاحية: تحويل النص إلى SQL؛ النماذج اللغوية الصغيرة (SLMs)؛ الذكاء الاصطناعي لقواعد البيانات؛ مساعدين ذكيين محليين.

Abstract

This thesis falls within the field of artificial intelligence applied to natural language database querying, specifically addressing the Text-to-SQL task. The main objective is to design and evaluate a local assistant based on Small Language Models (SLMs) capable of translating natural language queries into executable SQL statements, while ensuring performance suitable for resource-constrained environments.

The study provides an in-depth comparative evaluation of several language models executed locally via LM Studio, including SQL-specialized models (SQLCoder-7B, SLM-SQL-1.5B, Gemma Text-to-SQL), general-purpose models (Qwen 3.5, DeepSeek-R1-Distill-Qwen), as well as edge-optimized models designed for low resource consumption (Ministral 3B, IBM Granite Tiny). These models cover different sizes, architectures, and quantization strategies, enabling a fine-grained analysis of the trade-off between performance, accuracy, and computational cost.

The methodology is based on a reproducible experimental protocol involving multilingual SQL generation scenarios, including both simple and complex cases. Performance is evaluated according to several criteria, including syntactic correctness of generated queries, semantic validity, generalization capability, as well as CPU inference time.

The results show that SQL-specialized models achieve higher accuracy on structured queries, while some larger general-purpose models perform better in complex reasoning tasks. Edge models, although less accurate, stand out for their efficiency in terms of speed and resource consumption, making them suitable for embedded environments.

Finally, this study highlights the importance of model selection depending on the use case context and emphasizes the potential of hybrid architectures and distillation techniques to improve local Text-to-SQL systems.

Keywords : Text-to-SQL ; Small Language Models (SLMs) ; Artificial intelligence for databases ; Local intelligent assistants.

Table des matières

Table des figures

Liste des tableaux

Liste des Abréviations

Abréviation	Signification
AI	Artificial Intelligence (Intelligence Artificielle)
ANSI	American National Standards Institute
API	Application Programming Interface
ASC	Ascending
AVG	Average
BERT	Bidirectional Encoder Representations from Transformers
CPU	Central Processing Unit
COUNT	Count (Fonction d'agrégation)
DESC	Descending
DCL	Data Control Language
DDL	Data Definition Language
DML	Data Manipulation Language
FK	Foreign Key
FLOPs	Floating Point Operations
FP16	Floating Point 16-bit
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
IBM	International Business Machines
ISO	International Organization for Standardization
LLM	Large Language Models
LoRA	Low-Rank Adaptation
LSTM	Long Short-Term Memory
MAX	Maximum
MIN	Minimum
NL2SQL	Natural Language to SQL
NLP	Natural Language Processing
PK	Primary Key

Abréviation	Signification
RAG	Retrieval-Augmented Generation
RGPD	Règlement Général sur la Protection des Données
RNN	Recurrent Neural Networks
SGBD	Système de Gestion de Bases de Données
SLM	Small Language Models
SQL	Structured Query Language
SUM	Sum (Fonction d'agrégation)
T5	Text-to-Text Transfer Transformer
TCL	Transaction Control Language
VRAM	Video Random Access Memory

Introduction Générale

Introduction Générale

L'explosion du volume des données numériques au cours des dernières années a profondément modifié les méthodes de gestion, d'analyse et de prise de décision au sein des organisations. Les **bases de données relationnelles** constituent aujourd'hui l'un des principaux moyens de stockage et d'exploitation de ces données. Grâce à leur robustesse, leur flexibilité et leur large adoption, elles sont utilisées dans de nombreux domaines tels que les administrations publiques, les entreprises, les établissements de santé, les institutions financières ou encore les centres de recherche. Cependant, l'accès aux informations contenues dans ces bases repose généralement sur l'utilisation du langage **SQL** (Structured Query Language), dont la maîtrise nécessite des compétences techniques spécifiques.

Cette dépendance au **SQL** représente un obstacle important pour les utilisateurs non spécialistes qui souhaitent interroger les données ou comprendre les traitements effectués par une requête. Même lorsqu'une requête est correctement exécutée, son interprétation peut s'avérer difficile en raison de sa complexité syntaxique, de la présence de jointures multiples, de sous-requêtes ou d'opérations d'agrégation avancées. Dans ce contexte, le développement de systèmes capables de traduire automatiquement une question posée en **langage naturel** en une **requête SQL exécutable** apparaît comme une problématique de recherche particulièrement pertinente.

Le domaine du **Text-to-SQL** s'intéresse précisément à cette problématique. Son objectif est de traduire automatiquement une question formulée en **langage naturel** en une requête **SQL** exécutable. Une telle traduction permet non seulement de faciliter l'accès aux données sans maîtrise du SQL, mais également d'améliorer la transparence des systèmes de gestion de données, de simplifier l'interrogation des bases et d'assister les utilisateurs dans leurs activités d'analyse. Cette problématique s'inscrit dans le domaine plus large du **traitement automatique du langage naturel** (**NLP**) et bénéficie aujourd'hui des avancées réalisées dans le domaine des modèles de langage.

L'apparition des **Large Language Models (LLM)** a considérablement amélioré les capacités de compréhension et de génération de texte des systèmes d'intelligence artificielle. Des modèles tels que **GPT**, **Claude** ou **Gemini** démontrent des performances remarquables dans de nombreuses tâches linguistiques, notamment la traduction, le résumé, la génération de code ou encore l'assistance conversationnelle. Toutefois, malgré leur efficacité, ces modèles présentent plusieurs limitations lorsqu'ils doivent être intégrés dans des environnements locaux. Leur taille importante nécessite des ressources matérielles considérables, leur coût d'exploitation peut être élevé et leur utilisation via des services **cloud** soulève des questions liées à la **confidentialité** et à la **souveraineté des données**.

Face à ces limitations, les **Small Language Models (SLM)** apparaissent comme une al-

ternative particulièrement intéressante. Bien que plus compacts que les **LLM**, ces modèles conservent des capacités de raisonnement et de génération suffisantes pour de nombreuses applications professionnelles. Leur faible empreinte mémoire, leur rapidité d'exécution et leur aptitude à être déployés **localement** en font des candidats pertinents pour le développement d'applications intelligentes respectant les contraintes de **confidentialité** et de maîtrise des ressources. Cependant, les performances des **SLM** peuvent varier considérablement selon l'architecture adoptée et les mécanismes mis en œuvre pour enrichir leur compréhension du contexte.

Dans ce cadre, la présente recherche vise à étudier différentes approches fondées sur les **SLM** afin d'identifier les solutions les plus adaptées à la traduction de questions en **langage naturel** en requêtes **SQL** exécutables. Plusieurs architectures sont considérées. La première repose sur l'utilisation directe d'un **SLM** sans mécanisme complémentaire. La seconde enrichit le modèle à l'aide d'une approche **Retrieval-Augmented Generation (RAG)** permettant d'intégrer des informations contextuelles externes lors de la génération. La troisième s'appuie sur une architecture **Multi-Agent** dans laquelle plusieurs agents spécialisés collaborent pour analyser, contextualiser, générer et évaluer les réponses. Enfin, une quatrième approche, basée sur **PANDAS + SLM**, explore un paradigme différent dans lequel le modèle génère automatiquement du code **Python** pour manipuler des données stockées sous forme de **DataFrame** et répondre aux questions formulées en langage naturel.

L'objectif principal de ce mémoire est de réaliser une étude comparative approfondie de ces différentes approches afin d'identifier celle qui offre le meilleur compromis entre qualité des réponses, coût computationnel, simplicité d'intégration et aptitude au **déploiement local**. Cette étude constitue une étape essentielle dans la conception de **ASK-DATA**, une **application web** intelligente fonctionnant localement et destinée à faciliter l'interaction entre les utilisateurs et les données.

Pour atteindre cet objectif, une campagne expérimentale est menée sur le benchmark **Spider**, considéré comme l'une des références les plus utilisées dans la littérature pour l'évaluation des systèmes liés aux bases de données et au traitement du langage naturel. Les expérimentations sont réalisées sur un ensemble de **dix Small Language Models** représentatifs de différentes familles de modèles **open source**. Une première phase d'évaluation consiste à comparer les performances des modèles dans une configuration **SLM** pure sur processeur **(CPU)** puis sur processeur graphique **(GPU)**. Les résultats obtenus permettent ensuite de sélectionner les modèles les plus performants afin de poursuivre les expérimentations sur les autres architectures étudiées.

Les performances sont analysées selon plusieurs critères complémentaires, notamment la qualité des réponses générées, la fidélité des explications par rapport aux requêtes **SQL**, le **temps d'exécution**, la **consommation de ressources** matérielles et la faisabilité d'un **déploiement local**. Cette approche méthodologique permet non seulement de comparer objec-

tivement les différentes architectures, mais également d'identifier les avantages et les limites de chacune dans le contexte spécifique du projet **ASK-DATA**.

Les contributions principales de ce mémoire peuvent être résumées comme suit :

- réalisation d'une étude comparative de plusieurs **Small Language Models** pour la tâche **Text-to-SQL**.
- évaluation de différentes architectures d'exploitation des **SLM** : **SLM pur**, **SLM + RAG** et **Multi-Agent**.
- analyse d'une approche alternative basée sur la génération automatique de code **Python** à l'aide de **PANDAS + SLM**.
- étude des performances sur le benchmark **Spider** dans un contexte de **déploiement local**.
- identification de l'architecture la plus adaptée au développement de l'application web **ASK-DATA**.

Le présent mémoire est organisé en cinq chapitres principaux. Le premier chapitre présente les concepts fondamentaux relatifs aux modèles de langage, aux **Small Language Models** et aux systèmes **Text-to-SQL**, ainsi qu'un état de l'art des travaux existants. Le deuxième chapitre décrit les différentes approches **SLM** étudiées et analyse leurs caractéristiques théoriques. Le troisième chapitre détaille le protocole expérimental, les modèles évalués, les données utilisées ainsi que les métriques de performance retenues. Le quatrième chapitre présente les résultats expérimentaux obtenus, discute les performances des différentes architectures et identifie la solution la plus appropriée. Le cinquième chapitre présente la conception et le développement de l'application web **ASK-DATA**. Enfin, une conclusion générale synthétise les principaux apports du travail réalisé et présente les perspectives futures de recherche et de développement.

Chapitre 1

État de l'art sur les systèmes Text-to-SQL et les Small Language Models

1.1 Introduction

Ce chapitre présente les fondements théoriques et technologiques nécessaires à la compréhension des travaux réalisés dans ce mémoire. Il introduit d'abord les modèles de langage de grande taille (Large Language Models) et leurs principales caractéristiques, avant d'aborder les Small Language Models, qui constituent le cœur de cette étude. Les concepts fondamentaux liés aux bases de données relationnelles et au langage SQL sont ensuite présentés afin de contextualiser la problématique Text-to-SQL. Enfin, un état de l'art des approches existantes est réalisé, suivi d'une analyse critique permettant d'identifier les limites des travaux actuels et de justifier les contributions proposées dans ce mémoire.

1.2 Large Language Models (LLM)

Les **Large Language Models (LLM),** ou grands modèles de langage, représentent une classe de modèles d'intelligence artificielle fondés sur des réseaux de neurones profonds, entraînés sur des corpus textuels massifs et diversifiés. Leur objectif principal est de modéliser la distribution de probabilité des séquences de mots, leur permettant de comprendre, générer et manipuler le langage naturel avec un niveau de fluidité et de cohérence inédit.

1.2.1 Évolution des modèles de langage

L'histoire des modèles de langage a connu plusieurs ruptures technologiques majeures. Initialement dominés par des approches statistiques basées sur les n-grammes, limités par leur incapacité à capturer des dépendances à long terme, les modèles ont évolué vers des architectures neuronales récurrentes **(RNN)**, notamment les réseaux à mémoire à long terme **(LSTM) Hochreiter et Schmidhuber (1997)**. Bien que ces derniers aient amélioré la gestion des séquences, ils souffraient de problèmes de disparition du gradient et d'un traitement séquentiel empêchant la parallélisation.

L'avènement des représentations vectorielles denses **(Word Embeddings)**, telles que Word2Vec Mikolov et al.(2013), a marqué une première étape vers la sémantique contextuelle. Cependant, le véritable changement de paradigme est survenu en 2017 avec l'introduction de l'architecture Transformer, qui a rendu obsolètes les approches récurrentes en permettant un traitement parallèle des séquences et une modélisation efficace des dépendances à longue distance Vaswani et al. (2017). Cette évolution a été accompagnée par la mise en évidence des "lois d'échelle" (scaling laws), démontrant que la performance des modèles s'améliore de manière prévisible avec l'augmentation du nombre de paramètres, de la taille du jeu de données et de la puissance de calcul (Kaplan et al., 2020).

1.2.2 Architecture Transformer : fondements et principes

Le **Transformer** repose sur une architecture de type encodeur-décodeur (bien que la plupart des LLM génératifs modernes n'utilisent que la partie décodeur). Son innovation fondamentale réside dans l'abandon des boucles de récurrence au profit d'un mécanisme d'attention global. Cette architecture permet de traiter tous les tokens **(unités lexicales)** d'une séquence d'entrée simultanément, offrant une parallélisation massive lors de l'entraînement. Le modèle intègre également des mécanismes de normalisation de couche **(Layer Normalization)** et des connexions résiduelles **(Residual Connections)** pour stabiliser l'apprentissage des réseaux très profonds.

1.2.3 Principes de fonctionnement des LLM

Pré-entraînement auto-supervisé : La phase de pré-entraînement est le socle des LLM. Elle est dite "auto-supervisée" car elle ne nécessite pas d'annotations humaines. Le modèle apprend en résolvant une tâche de prédiction sur de vastes corpus non structurés. Deux paradigmes dominent : la modélisation du langage masqué **(Masked Language Modeling)**, où le modèle doit deviner des mots cachés dans une phrase **(ex : BERT)**, et la prédiction du token suivant **(Next-Token Prediction)**, où le modèle apprend à prédire le mot suivant dans une séquence, de gauche à droite **(ex : GPT)**. Ce processus permet au modèle d'acquérir des représentations riches de la grammaire, de la syntaxe et des connaissances factuelles du monde.

Mécanisme d'attention : Le cœur du Transformer est le mécanisme d'attention, et plus spécifiquement l'attention à têtes multiples **(Multi-Head Self-Attention)**. Pour chaque token, ce mécanisme calcule trois vecteurs : une requête **(Query)**, une clé **(Key)** et une valeur **(Value)**. En calculant le produit scalaire entre les requêtes et les clés de tous les tokens de la séquence, le modèle attribue un poids d'importance à chaque mot par rapport aux autres. Cela permet au modèle de "se concentrer" sur les parties pertinentes du contexte, indépendamment de leur distance positionnelle dans la phrase, résolvant ainsi le problème des dépendances à long terme.

Génération autoregressive du texte : Lors de la phase d'inférence, les LLM de type décodeur génèrent du texte de manière autoregressive. Le modèle prédit la distribution de probabilité du prochain token conditionnée par tous les tokens précédents : $P(w_t|w_1, w_2, ..., w_{t-1})$. Le token ayant la probabilité la plus élevée (ou échantillonné selon une stratégie comme le Top-k ou le Nucleus sampling / Top-p) est ajouté à la séquence, et le processus se répète jusqu'à la génération d'un token de fin de séquence ou l'atteinte d'une longueur maximale.

1.2.4 Principaux modèles LLM récents

Le paysage des **LLM** est en évolution rapide. Parmi les modèles fondateurs et les plus influents, on retrouve :

• La série **GPT** (Generative Pre-trained Transformer) d'OpenAI Brown et al. (2020), qui a popularisé l'approche decoder-only et démontré les capacités few-shot.

• **BERT** Devlin et al. (2018) de Google, référence des modèles encoder-only optimisés pour la compréhension et la classification.

• **LLaMA / Llama 2 / Llama 3** Touvron et al. (2023) de Meta, qui ont marqué un tournant en démontrant que des modèles de taille plus modérée, entraînés sur des données de haute qualité, pouvaient rivaliser avec des modèles massifs, tout en étant partiellement ouverts.

• **Mistral** Jiang et al. (2023), un modèle européen open-weight notable pour son architecture optimisée (comme l'attention à fenêtre glissante) offrant un excellent rapport performance/efficacité.

1.2.5 Avantages des LLM

Capacités de compréhension contextuelle : Contrairement aux anciennes représentations statiques (comme Word2Vec), les LLM génèrent des embeddings contextuels. Un même mot aura une représentation vectorielle différente selon la phrase dans laquelle il apparaît, permettant une désambiguïsation sémantique fine et une compréhension des nuances, de l'ironie ou des références implicites.

Capacités de génération de texte : Les LLM produisent un langage naturel d'une fluidité remarquable. Ils sont capables de maintenir la cohérence thématique sur de longs passages, d'adapter leur style, leur ton et leur format de sortie **(résumé, code, poésie, rapport technique)** en fonction des instructions fournies.

Polyvalence des tâches : Grâce à l'apprentissage en contexte (In-Context Learning), un LLM pré-entraîné peut accomplir une multitude de tâches (traduction, analyse de sentiment, extraction d'entités, raisonnement logique) sans nécessiter de ré-entraînement (fine-tuning) spécifique. Il suffit de lui fournir quelques exemples (few-shot prompting) ou une instruction claire (zero-shot prompting) pour qu'il adapte son comportement (Wei et al., 2022).

1.2.6 Limites des LLM pour les déploiements locaux

Malgré leurs performances, l'exploitation des LLM en environnement local (on-premise) se heurte à des contraintes techniques et opérationnelles majeures, justifiant souvent le recours à des solutions cloud ou à des techniques d'optimisation spécifiques.

Coût computationnel élevé : La complexité algorithmique du mécanisme d'attention standard est quadratique ($O(N^2)$) par rapport à la longueur de la séquence (N). Cela rend l'inférence et l'entraînement extrêmement coûteux en opérations en virgule flottante **(FLOPs)**, en particulier pour les contextes longs.

Exigences matérielles importantes : Le déploiement d'un LLM nécessite de charger l'intégralité de ses paramètres en mémoire vive graphique **(VRAM)**. Par exemple, un modèle de 70 milliards de paramètres en précision 16 bits **(FP16)** nécessite environ 140 Go de **VRAM**, ce qui excède la capacité d'une carte graphique grand public et impose l'utilisation de clusters de serveurs équipés de GPU professionnels **(ex : NVIDIA A100/H100)**, même avec des techniques de quantification.

Latence d'inférence : La génération autoregressive est intrinsèquement séquentielle : chaque nouveau token dépend du calcul des précédents. Cette opération est fortement limitée par la bande passante mémoire (memory-bound) plutôt que par la puissance de calcul brute, entraînant une latence perceptible (temps jusqu'au premier token et vitesse de génération en tokens/seconde), inacceptable pour certaines applications en temps réel.

Problématiques de confidentialité des données : L'utilisation **d'API cloud** pour interroger des LLM propriétaires implique l'envoi de données d'entreprise ou personnelles à des tiers. Cela entre en conflit avec des réglementations strictes comme le **RGPD** en Europe,

ou des politiques de sécurité internes (secteurs de la santé, de la finance ou de la défense), rendant le traitement local des données impératif, mais techniquement difficile.

Dépendance aux services cloud : Le recours aux LLM hébergés crée une dépendance vis-à-vis de la connectivité réseau, de la disponibilité des services du fournisseur (vendor lock-in) et de la volatilité des coûts d'API, qui sont généralement facturés au nombre de tokens traités, rendant les coûts imprévisibles à grande échelle.

1.3 Small Language Models (SLM)

1.3.1 Définition et positionnement des SLM

Les **Small Language Models (SLM)** sont définis comme des architectures légères conçues pour l'efficacité, l'adaptabilité et le déploiement dans des environnements aux ressources limitées. Contrairement aux grands modèles, un SLM contient généralement **moins de 10 milliards de paramètres.** Ils se positionnent comme une alternative pratique et durable face aux préoccupations croissantes concernant la consommation d'énergie, la latence et les coûts élevés associés aux modèles de taille massive.

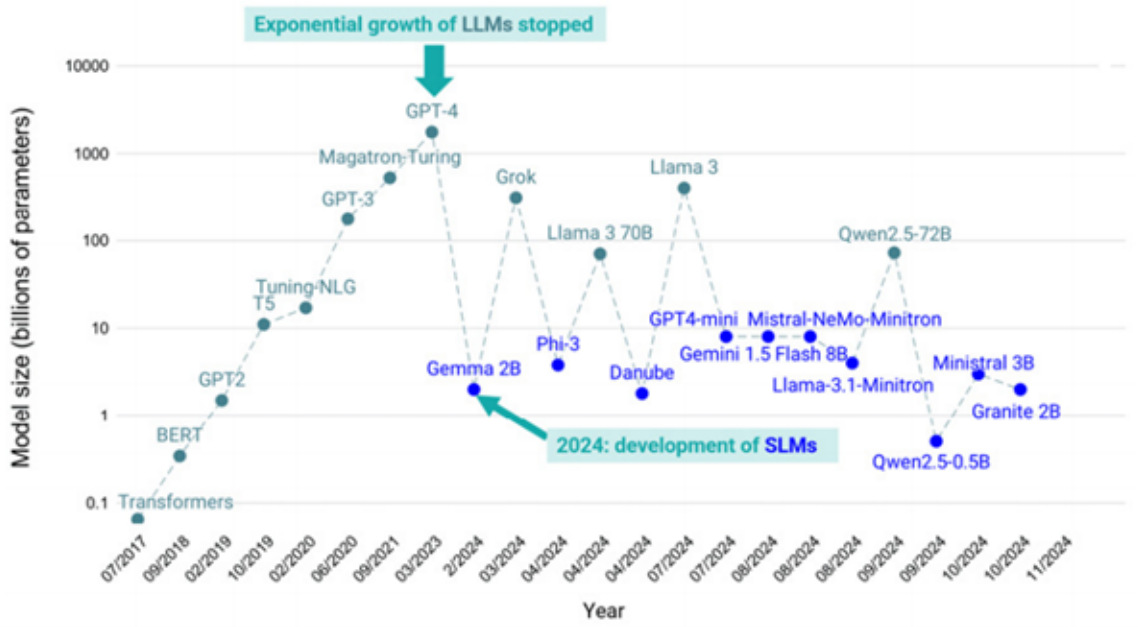


FIG. 1.1 : Chronologie de l'émergence des modèles de langage Source (Tham et al., 2025).

1.3.2 Caractéristiques des Small Language Models

Taille des modèles

Les SLM privilégient la compacité, avec des exemples notables tels que DistilBERT (66M), TinyBERT (14,5M) ou Phi-2 (2,7B).

Consommation mémoire

Ils sont optimisés pour l'efficacité de la mémoire, réduisant considérablement la charge de calcul par rapport aux LLM.

Efficacité énergétique

Leur conception permet un déploiement efficace en périphérie (edge computing) avec une consommation d'énergie réduite.

Architectures principales des SLM

Bien que basés sur l'architecture **Transformer,** les SLM utilisent des configurations compactes avec une profondeur et une largeur de réseau réduites. Ils peuvent adopter des structures de type **encodeur seul** (comme BERT), **décodeur seul** (comme les modèles de style GPT) ou **encodeur-décodeur** (comme T5) selon les besoins de la tâche.

1.3.3 Techniques d'optimisation des SLM

Quantification

Cette technique (ex : 8-bit, 4-bit) réduit les besoins en mémoire et en calcul après l'entraînement.

Distillation de connaissances

Un modèle "étudiant" plus petit est entraîné pour imiter les performances d'un modèle "enseignant" plus grand, comme c'est le cas pour DistilBERT ou TinyBERT.

Fine-tuning léger

L'adaptation de bas rang **(LoRA)** permet un ajustement précis et efficace des paramètres en injectant des matrices entraînables dans des modèles de base gelés.

1.3.4 Avantages des SLM pour les systèmes locaux

Déploiement sur matériel limité :

Les SLM peuvent s'exécuter sur des **CPU,** des smartphones ou des systèmes embarqués.

Réduction des coûts :

Ils offrent des coûts de formation et d'inférence bas, rendant leur déploiement à grande échelle économiquement viable.

Protection de la confidentialité :

Ils facilitent l'inférence locale privée, ce qui est idéal pour les déploiements sensibles à la sécurité des données.

Fonctionnement hors ligne :

Leur légèreté permet des applications NLP performantes sans dépendre d'une connexion cloud.

1.3.5 Défis et limitations des SLM

Réduction des capacités de raisonnement :

En raison de leur taille, ils sacrifient souvent une partie des capacités de raisonnement généraliste et des comportements émergents propres aux LLM.

Limitation de la fenêtre de contexte :

Leur fenêtre de contexte est généralement limitée à une plage de **2k à 8k tokens**, contre 128k pour certains modèles massifs.

Sensibilité à la qualité des prompts :

Ils peuvent se révéler fragiles et moins contrôlables en raison d'une conscience contextuelle réduite, ce qui nécessite souvent un réglage fin pour obtenir des résultats comparables.

Difficultés sur les tâches complexes :

Ils ont tendance à être moins performants sur des tâches de résolution de problèmes multi-étapes ou de généralisation "zero-shot".

1.4 Bases de données relationnelles et langage SQL

1.4.1 Concepts fondamentaux des bases de données relationnelles

Le modèle relationnel, introduit par Edgar F. Codd en 1970, constitue le fondement théorique de la majorité des systèmes de gestion de bases de données (SGBD) modernes. Il repose sur des concepts mathématiques issus de la théorie des ensembles et de la logique des prédicats du premier ordre (Codd, 1970).

Tables et attributs : Dans le modèle relationnel, une relation est représentée logiquement par une table. Une table est un ensemble non ordonné de n-uplets (ou tuples), où chaque n-uplet représente un enregistrement ou une entité du monde réel. Les colonnes de la table sont appelées attributs. Chaque attribut possède un nom unique au sein de la table et est défini sur un domaine de valeurs spécifique (par exemple, un entier, une chaîne de caractères ou une date), garantissant ainsi l'atomicité des données (première forme normale) (Date, 2003).

Clés primaires et étrangères : L'intégrité des données est assurée par deux types de contraintes fondamentales :

- **La clé primaire (Primary Key - PK) :** Il s'agit d'un attribut, ou d'un ensemble minimal d'attributs, dont la valeur identifie de manière unique chaque n-uplet d'une relation. Elle ne peut contenir de valeur nulle (contrainte d'entité).
- **La clé étrangère (Foreign Key - FK) :** Il s'agit d'un attribut (ou groupe d'attributs) dans une table qui fait référence à la clé primaire d'une autre table (ou de la même table). Elle assure l'intégrité référentielle, empêchant l'existence de références orphelines et matérialisant le lien sémantique entre les entités (Melton & Simon, 2001).

Relations entre tables : Les relations entre les tables sont définies par leur cardinalité, qui décrit le nombre d'occurrences d'une entité pouvant être associées à une occurrence d'une autre entité. On distingue principalement trois types de relations :

- **Un à un (1:1) :** Une occurrence de l'entité A est associée à au plus une occurrence de l'entité B, et vice-versa.
- **Un à plusieurs (1:N) :** Une occurrence de l'entité A peut être associée à plusieurs occurrences de l'entité B, mais une occurrence de B n'est associée qu'à une seule occurrence de A. C'est la relation la plus courante, implémentée via une clé étrangère dans la table du côté "plusieurs".
- **Plusieurs à plusieurs (M :N) :** Une occurrence de A peut être associée à plusieurs occurrences de B, et inversement. Ce type de relation nécessite la création d'une table de jointure (ou table d'association) pour être résolu en deux relations 1:N.

1.4.2 Langage SQL

Présentation générale : Le SQL (Structured Query Language) est le langage standardisé pour la définition, la manipulation et le contrôle des données dans les SGBD relationnels. Développé initialement par IBM (sous le nom de SEQUEL) dans les années 1970 par Chamberlin et Boyce, il a été normalisé pour la première fois par l'ANSI en 1986, puis par l'ISO en 1987. Le SQL est un langage déclaratif : l'utilisateur spécifie ce qu'il veut obtenir (le résultat), et non comment l'obtenir (l'algorithme d'exécution), ce dernier étant déterminé par l'optimiseur de requêtes du SGBD (Elmasri & Navathe, 2015).

Catégories des commandes SQL : Le standard SQL divise ses instructions en plusieurs sous-langages distincts :

- **LDD (Langage de Définition de Données) / DDL :** Permet de définir et de modifier la structure des objets de la base (ex : `CREATE`, `ALTER`, `DROP`).

- **LMD (Langage de Manipulation de Données) / DML :** Permet d'interroger et de modifier le contenu des tables (ex : `SELECT`, `INSERT`, `UPDATE`, `DELETE`).
- **LCD (Langage de Contrôle de Données) / DCL :** Gère les droits d'accès et les permissions (ex : `GRANT`, `REVOKE`).
- **LCT (Langage de Contrôle de Transaction) / TCL :** Gère l'intégrité des transactions (ex : `COMMIT`, `ROLLBACK`, `SAVEPOINT`).

1.4.3 Structure d'une requête SQL

Une requête SQL de sélection (`SELECT`) suit une syntaxe rigide, bien que l'ordre d'exécution logique par le SGBD diffère de l'ordre d'écriture syntaxique. L'ordre logique d'évaluation est généralement : `FROM` → `WHERE` → `GROUP BY` → `HAVING` → `SELECT` → `ORDER BY` (Garcia-Molina et al., 2008).

- **Clause SELECT :** Spécifie les attributs (colonnes) ou les expressions à projeter dans le résultat final. Elle peut inclure le mot-clé `DISTINCT` pour éliminer les doublons.
- **Clause FROM :** Identifie les tables sources (relations) sur lesquelles l'interrogation porte. C'est la première étape évaluée, générant un produit cartésien initial (ou une jointure) des tables mentionnées.
- **Clause WHERE :** Applique un prédicat de filtrage au niveau des lignes (n-uplets) avant toute agrégation. Seules les lignes satisfaisant la condition booléenne sont conservées.
- **GROUP BY et HAVING :** La clause `GROUP BY` partitionne les lignes résultantes en groupes basés sur les valeurs d'un ou plusieurs attributs. La clause `HAVING` agit ensuite comme un filtre, mais au niveau des groupes (et non des lignes individuelles), permettant d'utiliser des fonctions d'agrégation dans sa condition.
- **Clause ORDER BY :** Effectue le tri final du jeu de résultats selon un ou plusieurs attributs, par ordre croissant (`ASC`) ou décroissant (`DESC`).

1.4.4 Complexité des requêtes SQL

La puissance du SQL réside dans sa capacité à exprimer des interrogations complexes, mais cette expressivité s'accompagne d'une augmentation de la complexité algorithmique et cognitive.

Requêtes multi-tables : L'interrogation simultanée de plusieurs tables est rendue possible par les opérations de jointure. La complexité réside dans la nécessité de spécifier correctement les conditions de jointure pour éviter les produits cartésiens involontaires, qui dégradent exponentiellement les performances.

Jointures complexes : Au-delà de la jointure interne (`INNER JOIN`), le SQL offre des mécanismes avancés :

- **Jointures externes (`LEFT/RIGHT/FULL OUTER JOIN`) :** Préservent les lignes d'une table même si aucune correspondance n'est trouvée dans l'autre table, en complétant par des valeurs `NULL`.
- **Auto-jointures (`SELF JOIN`) :** Une table est jointe à elle-même, utile pour modéliser des hiérarchies (ex : un employé et son manager dans la même table).

Sous-requêtes imbriquées : Une sous-requête est une requête `SELECT` incluse dans une autre requête. Elles peuvent être :

- **Non corrélées :** Exécutées une seule fois, indépendamment de la requête principale (souvent utilisées avec `IN` ou `EXISTS`).
- **Corrélées :** Exécutées répétitivement pour chaque ligne traitée par la requête externe, car elles font référence à des variables de cette dernière. Bien que puissantes, elles peuvent engendrer des coûts de calcul élevés si elles ne sont pas optimisées par le SGBD.

Fonctions d'agrégation : Les fonctions telles que `COUNT()`, `SUM()`, `AVG()`, `MIN()` et `MAX()` opèrent sur un ensemble de valeurs pour retourner une valeur scalaire unique. Leur utilisation correcte nécessite une compréhension fine de l'interaction entre les attributs de regroupement (`GROUP BY`) et les attributs agrégés.

Requêtes analytiques : Introduites dans le standard SQL :2003, les fonctions de fenêtrage (Window Functions) représentent un saut majeur en complexité et en expressivité. Contrairement aux fonctions d'agrégation classiques qui réduisent le nombre de lignes, les fonctions analytiques (ex : `ROW_NUMBER()`, `RANK()`, `LEAD()`, `LAG()`) calculent une valeur pour chaque ligne en se basant sur un ensemble de lignes liées (la "fenêtre"), définie par la clause `OVER (PARTITION BY ... ORDER BY ...)`. Elles permettent d'effectuer des calculs complexes (moyennes mobiles, classements) sans recourir à des auto-jointures coûteuses.

1.4.5 Difficultés d'interprétation des requêtes SQL

L'interprétation et la maintenance de requêtes SQL complexes posent plusieurs défis, tant pour les développeurs que pour les systèmes d'optimisation :

1. **Décalage cognitif (Syntaxe vs Exécution) :** L'ordre d'écriture des clauses (`SELECT` en premier) ne correspond pas à l'ordre d'exécution logique (`FROM` en premier). Cela crée une charge cognitive pour le développeur qui doit mentalement réorganiser la requête pour en comprendre la sémantique, notamment concernant la portée des alias de colonnes.

2. **Ambiguïté sémantique :** L'absence de commentaires, l'utilisation d'alias non descriptifs (ex : `t1`, `t2`) ou l'imbrication excessive de sous-requêtes rendent la logique métier sous-jacente opaque.

3. **Complexité de l'optimisation :** Pour le SGBD, une requête complexe (avec multiples jointures, sous-requêtes corrélées et fonctions de fenêtrage) génère un espace de plans d'exécution gigantesque. L'interprétation de ces plans (via `EXPLAIN` ou `EXPLAIN ANALYZE`) nécessite une expertise approfondie pour identifier les goulots d'étranglement (scans de table complets, jointures de hachage non optimales, etc.).

4. **Gestion des valeurs NULL :** La logique tri-valuée du SQL (Vrai, Faux, Inconnu) rend l'interprétation des conditions `WHERE` ou des jointures contre-intuitive lorsque des valeurs `NULL` sont présentes, source fréquente de bugs logiques silencieux (Date, 2003).

1.5 Présentation du problème Text-to-SQL

1.5.1 Définition

Le **Text-to-SQL** (ou NL2SQL : Natural Language to SQL) est une tâche de traitement automatique du langage naturel (NLP) qui consiste à convertir automatiquement une requête exprimée en langage naturel en une requête SQL exécutable sur une base de données relationnelle. Son objectif principal est de permettre aux utilisateurs non experts en bases de données d'accéder aux informations stockées sans avoir à maîtriser la syntaxe SQL.

Question en langage naturel

« Trouver toutes les villes ayant une population supérieure à un million aux États-Unis »

↓ **Système Text-to-SQL** ↓

```
SELECT city_name FROM Cities
WHERE population > 1000000 AND country = 'United States';
```

Le problème Text-to-SQL se situe à l'intersection de plusieurs domaines de recherche, notamment le traitement automatique du langage naturel, l'intelligence artificielle, l'apprentissage automatique et les systèmes de gestion de bases de données. Il constitue un élément clé dans le développement d'interfaces conversationnelles permettant l'interrogation intuitive des données.

Applications :

Les systèmes Text-to-SQL trouvent aujourd'hui de nombreuses applications dans les environnements académiques et industriels.

- **Intelligence décisionnelle (Business Intelligence) :** Les entreprises stockent de grandes quantités de données dans des entrepôts de données relationnels. Les systèmes Text-to-SQL permettent aux décideurs d'interroger directement ces données en langage naturel afin d'obtenir des rapports, des statistiques ou des indicateurs de performance sans dépendre d'analystes spécialisés.
- **Interfaces conversationnelles et assistants intelligents :** Les chatbots et assistants virtuels peuvent intégrer des capacités Text-to-SQL afin de répondre à des questions portant sur des bases de données internes. L'utilisateur interagit alors de manière naturelle tandis que le système traduit automatiquement ses requêtes en SQL.
- **Analyse de données et aide à la décision :** Les chercheurs, analystes et gestionnaires peuvent explorer rapidement des ensembles de données complexes à travers des questions formulées en langage naturel, facilitant ainsi l'extraction de connaissances et la prise de décision.
- **Applications scientifiques et médicales :** Dans des domaines spécialisés tels que la recherche scientifique, la santé ou l'astronomie, les systèmes Text-to-SQL permettent d'interroger des bases de données complexes contenant des millions d'enregistrements sans nécessiter une expertise approfondie en SQL.
- **Gestion de données d'entreprise :** Les systèmes modernes sont également utilisés dans les environnements professionnels où les bases de données peuvent contenir des

centaines de tables et plusieurs milliers de colonnes. Les nouvelles générations de benchmarks, comme Spider 2.0 ou BIRD, reflètent précisément ces scénarios réels.

Enjeux de recherche :

Malgré les progrès importants réalisés ces dernières années, le développement de systèmes Text-to-SQL performants demeure un défi scientifique majeur. Plusieurs problématiques de recherche restent ouvertes.

a) **Compréhension du langage naturel :** Les requêtes formulées par les utilisateurs peuvent être ambiguës, incomplètes ou exprimées de multiples façons. Le système doit être capable d'interpréter correctement l'intention de l'utilisateur et d'identifier les informations pertinentes dans la question.

b) **Correspondance avec le schéma de la base de données :** L'une des principales difficultés réside dans le *Schema Linking*, c'est-à-dire l'identification correcte des tables, colonnes et relations concernées par la requête. Cette étape est essentielle pour générer une requête SQL valide et précise.

c) **Génération de requêtes SQL complexes :** Les requêtes réelles impliquent souvent :

des jointures entre plusieurs tables, des agrégations (`COUNT`, `SUM`, `AVG`, etc.), des sous-requêtes imbriquées , des clauses `GROUP BY` et `HAVING` , des opérations ensemblistes.

La génération correcte de telles requêtes nécessite des capacités avancées de raisonnement et de compréhension du schéma relationnel.

d) **Généralisation à de nouveaux domaines :** Un système Text-to-SQL doit être capable de fonctionner sur des bases de données jamais vues durant l'entraînement. Cette capacité de généralisation constitue aujourd'hui l'un des principaux critères d'évaluation des modèles modernes.

e) **Robustesse et précision :** Les modèles doivent être capables de résister aux variations linguistiques, aux synonymes, aux erreurs de formulation et aux schémas complexes. Des benchmarks récents comme Dr.Spider ont montré que les systèmes actuels restent sensibles à de nombreuses perturbations.

f) **Déploiement local et confidentialité :** L'utilisation croissante des *Large Language Models* (LLM) améliore considérablement les performances des systèmes Text-to-SQL. Cependant, ces modèles nécessitent souvent d'importantes ressources de calcul et s'appuient sur des services cloud externes, ce qui soulève des questions de coût, de latence et de confidentialité des données. Ces contraintes motivent le développement de solutions basées sur des *Small Language Models* (SLM) capables d'être exécutés localement tout en offrant un niveau de performance satisfaisant.

g) **Explicabilité et confiance :** Dans les applications critiques, il est important de comprendre pourquoi une requête SQL particulière a été générée. Les travaux récents cherchent donc à intégrer des mécanismes de raisonnement explicite et de validation afin d'améliorer la transparence et la fiabilité des systèmes Text-to-SQL.

1.6 Travaux existants sur Text-to-SQL

1.6.1 Introduction

Le problème **Text-to-SQL** consiste à traduire automatiquement une requête exprimée en langage naturel vers une requête SQL exécutable sur une base de données relationnelle (voir Figure 1.2). Cette tâche vise à permettre aux utilisateurs non spécialistes des bases de données d'interroger directement les données sans maîtriser le langage SQL. Depuis plusieurs années, **Text-to-SQL** constitue un domaine de recherche majeur à l'intersection du traitement automatique du langage naturel (**NLP**) et des systèmes de gestion de bases de données.

L'évolution des systèmes **Text-to-SQL** peut être divisée en plusieurs générations : les approches fondées sur des règles, les modèles neuronaux de type **Seq2Seq**, les modèles pré-entraînés (**PLM**), puis plus récemment les approches basées sur les Large Language Models (**LLM**).

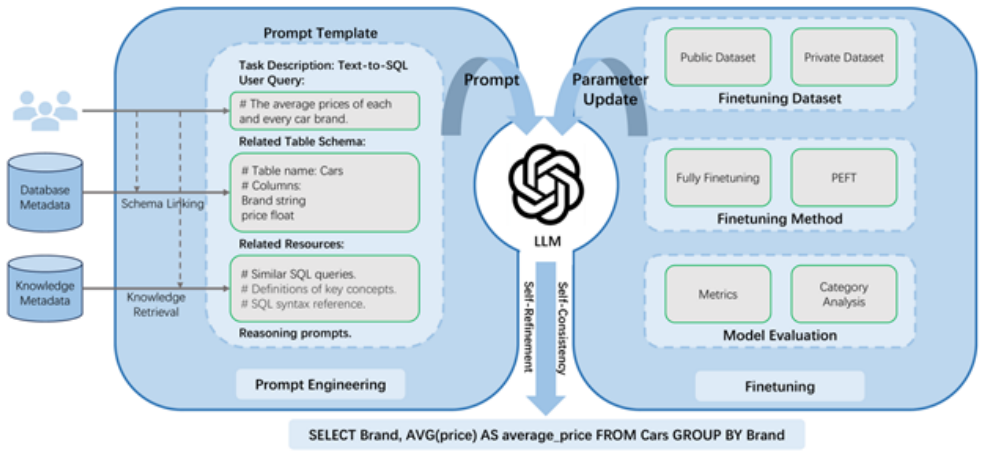


FIG. 1.2 : Cadre d'utilisation des LLM dans la conversion de texte en SQL source (Shi et al., 2025).

1.6.2 Approches basées sur des règles et des modèles

Les premiers systèmes Text-to-SQL reposaient principalement sur des règles manuelles et des modèles prédéfinis (template-based systems). Ces méthodes associaient des structures linguistiques spécifiques à des requêtes SQL prédéfinies. Bien qu'elles offrent une bonne

précision dans des domaines restreints, elles nécessitent un important travail d'ingénierie manuelle et présentent une faible capacité de généralisation à de nouveaux schémas de bases de données.

Ces limitations ont conduit à l'apparition de systèmes plus flexibles basés sur l'apprentissage automatique.

Approches neuronales Seq2Seq :

Avec l'essor du Deep Learning, les architectures Sequence-to-Sequence **(Seq2Seq)** sont devenues la principale approche pour le problème Text-to-SQL. Ces modèles apprennent directement une correspondance entre la requête en langage naturel et la requête SQL cible, sans nécessiter de règles explicites.

Parmi les travaux les plus influents figurent Shi et al. (2025) :SQLNet, Seq2SQL, IRNet, HydraNet, RyanSQL, RESDSQL, ISESL-SQL.

Ces modèles utilisent généralement un encodeur pour représenter la question et le schéma de la base de données, ainsi qu'un décodeur chargé de générer la requête SQL token par token.

Malgré leurs performances élevées, ces approches nécessitent un entraînement supervisé important et rencontrent souvent des difficultés de généralisation lorsqu'elles sont appliquées à de nouveaux domaines ou à des schémas non vus pendant l'entraînement.

Utilisation des modèles de langage pré-entraînés (PLM) :

L'étape suivante a consisté à intégrer des modèles de langage pré-entraînés tels que **BERT** afin d'améliorer la compréhension sémantique des requêtes utilisateurs et des schémas relationnels. Grâce à l'apprentissage sur de vastes corpus textuels, ces modèles ont permis d'améliorer significativement les performances des systèmes Text-to-SQL.

Les **PLM** ont constitué l'état de l'art avant l'apparition des grands modèles de langage modernes. Cependant, ils restent fortement dépendants de procédures de fine-tuning spécifiques et nécessitent des ensembles de données annotés de grande taille.

Émergence des Large Language Models :

L'arrivée des Large Language Models **(LLM)** tels que **GPT** a profondément transformé le domaine Text-to-SQL. Grâce à leurs capacités émergentes de compréhension du langage, de raisonnement et d'apprentissage en contexte (In-Context Learning), ces modèles peuvent générer des requêtes SQL complexes à partir de simples instructions textuelles.

Selon les travaux récents, les approches basées sur les LLM se répartissent principalement en deux catégories Shi et al. (2025) :

a) **Prompt Engineering :** Cette approche exploite directement les capacités du modèle pré-entraîné sans modification de ses paramètres. La qualité du résultat dépend fortement de la conception du prompt fourni au modèle. Les techniques les plus utilisées incluent : Zero-shot prompting, Few-shot prompting, Chain-of-Thought (CoT), Retrieval-Augmented Generation (RAG), Self-consistency et auto-correction.

Ces méthodes présentent l'avantage de ne nécessiter aucun réentraînement du modèle, mais leur efficacité dépend fortement du modèle utilisé et de la qualité du prompt.

b) **Fine-tuning des LLM :** La seconde catégorie consiste à spécialiser un LLM pré-entraîné sur des jeux de données Text-to-SQL. Cette stratégie permet d'améliorer les performances sur des domaines spécifiques mais nécessite davantage de ressources de calcul et de données annotées.

Techniques avancées utilisées dans les systèmes Text-to-SQL modernes :

Les systèmes récents intègrent plusieurs mécanismes complémentaires afin d'améliorer la qualité des requêtes générées.

a) **Schema Linking :** Le Schema Linking vise à identifier les tables et colonnes pertinentes dans le schéma de la base de données à partir de la question utilisateur. Cette étape réduit le bruit informationnel et améliore la précision de génération SQL.

b) **Raisonnement décomposé :** Plusieurs travaux utilisent des stratégies de raisonnement telles que : Chain-of-Thought (CoT), Least-to-Most Reasoning, Décomposition hiérarchique des tâches, Systèmes multi-agents.

Ces approches décomposent les requêtes complexes en sous-problèmes plus simples avant de générer la requête SQL finale.

c) **Post-traitement et auto-correction :** Après la génération SQL, certaines méthodes réalisent une phase de validation en utilisant : Les messages d'erreur du **SGBD**, la vérification syntaxique, la comparaison de plusieurs requêtes candidates, des mécanismes d'auto-correction.

Ces stratégies permettent d'améliorer la robustesse du système et de réduire les erreurs de génération.

Limites des approches actuelles :

Malgré les progrès récents, plusieurs défis subsistent :

- Dépendance à des modèles de très grande taille .
- Coûts élevés d'inférence .
- Exigences matérielles importantes .
- Problèmes de confidentialité liés à l'utilisation de services cloud .
- Difficultés de déploiement dans des environnements locaux ou embarqués .
- Performances encore limitées sur les bases de données complexes du monde réel.

Ces limitations motivent le développement de solutions basées sur des Small Language Models (SLM) capables d'être exécutés localement tout en conservant des performances acceptables pour les tâches Text-to-SQL.

L'évolution des systèmes Text-to-SQL montre une transition progressive des approches basées sur des règles vers des modèles neuronaux, puis vers les modèles de langage de grande taille. Les LLM ont considérablement amélioré les performances grâce à leurs capacités de compréhension et de raisonnement. Toutefois, leur coût de déploiement, leurs exigences matérielles et les contraintes de confidentialité limitent leur utilisation dans de nombreux contextes réels. Ces observations justifient l'intérêt croissant porté aux Small Language Models et constituent la motivation principale du présent travail de recherche visant à développer un système Text-to-SQL local.

1.7 Avancées récentes dans l'application des Small Language Models au Text-to-SQL

1.7.1 Définition et positionnement des SLM

Le Text-to-SQL reste un défi majeur en raison de la nécessité de lier correctement les questions aux schémas de base de données (*schema linking*) et de gérer la complexité des requêtes. Les SLM doivent non seulement comprendre la syntaxe SQL mais aussi la structure logique des jointures et des filtres.

Les SLM se définissent par une taille réduite, souvent comprise entre 0,5B et 1,5B paramètres. Bien qu'ils aient initialement des capacités de raisonnement logique plus limitées que les LLM, des techniques de post-entraînement avancées permettent désormais d'explorer leur plein potentiel pour des tâches spécialisées comme le Text-to-SQL.

1.7.2 Techniques d'optimisation des SLM

Post-entraînement et Raisonnement Duan et Wang, (2024) : L'utilisation de jeux de données synthétiques de haute qualité (comme SynSQL-2.5M) et l'intégration de *Chain-of-Thought* **(CoT)** via des balises `<think>` permettent aux SLM de décomposer les problèmes SQL complexe en étapes de raisonnement logiques.

Fine-tuning et Apprentissage par Renforcement (RL) Guo et al. (2025) : Le framework SLM-SQL démontre l'efficacité d'une approche en deux temps : un fine-tuning supervisé **(SFT)** pour la génération de SQL, suivi d'un post-entraînement par renforcement utilisant l'algorithme **GRPO** (*Group Relative Policy Optimization*). Le système de récompense repose sur la précision d'exécution **(REX)**.

- **Inférence par auto-correction (CSC-SQL)** Efimov (2025) : Une technique clé, la *Corrective Self-Consistency* (CSC), améliore la fiabilité. Elle consiste à générer plusieurs échantillons de requêtes, à voter sur les résultats d'exécution, puis à utiliser

un modèle de révision ("*merge revision model*") pour corriger les éventuelles incohérences. L'augmentation du budget de calcul au moment de l'inférence (nombre d'échantillons) est directement corrélée à l'amélioration de la précision.

1.7.3 Performance des architectures SLM-SQL :

Les recherches récentes sur des modèles comme **Qwen2.5-Coder-1.5B** montrent des résultats impressionnants, atteignant **67,08%** de précision d'exécution **(EX)** sur le benchmark BIRD. Ce niveau de performance permet aux SLM de surpasser des modèles bien plus grands (7B, 15B ou même 32B dans certains cas) et de rivaliser avec des solutions basées sur **GPT-4** (Sheng & Shuai, 2025).

1.7.4 Analyse critique de la littérature

1. **Comparaison et Généralisation :** Les méthodes SLM-SQL ont prouvé leur capacité de généralisation en obtenant des scores élevés sur le dataset **Spider** (jusqu'à **79,06%** EX pour un modèle 1,5B), même sans entraînement spécifique par renforcement sur ce dernier.

2. **Compromis Coût/Performance :** Les études montrent qu'un modèle de 1,5B sur un matériel local **(type NVIDIA 4090D)** peut traiter une question pour un coût dérisoire **($0,00046)**, offrant un équilibre optimal entre performance et efficience pour les entreprises.

1.8 Lacunes identifiées et positionnement du mémoire

1.8.1 Lacunes identifiées et positionnement du mémoire

L'analyse de la littérature révèle que les progrès récents des systèmes Text-to-SQL ont été largement portés par l'émergence des Large Language Models (LLM). Des modèles tels que GPT-4, Claude ou Gemini ont permis d'atteindre des niveaux de performance sans précédent sur les principaux benchmarks du domaine. Cependant, malgré ces avancées, plusieurs limitations subsistent lorsqu'il s'agit de déployer ces solutions dans des environnements réels nécessitant confidentialité, maîtrise des coûts et exécution locale. L'étude des travaux existants met en évidence plusieurs lacunes scientifiques qui motivent le présent mémoire.

1.8.2 Besoin d'évaluations comparatives des SLM

La majorité des travaux récents se concentrent sur l'utilisation de modèles de très grande taille comptant plusieurs dizaines ou centaines de milliards de paramètres. Les évaluations

publiées privilégient généralement les performances absolues obtenues par les LLM propriétaires tels que GPT-4, au détriment des modèles plus compacts.

Parallèlement, l'émergence des Small Language Models (SLM) tels que Llama 3, Phi, Gemma, Qwen ou Mistral ouvre de nouvelles perspectives pour les applications nécessitant un déploiement local. Néanmoins, les études comparatives systématiques de ces modèles dans le contexte spécifique du Text-to-SQL restent relativement limitées.

Plusieurs questions demeurent ouvertes :

- Quels SLM offrent le meilleur compromis entre précision et consommation de ressources ?
- Quel est l'impact de la taille du modèle sur la qualité des requêtes SQL générées ?
- Dans quelles conditions un SLM peut-il rivaliser avec un LLM plus volumineux ?
- Quels sont les modèles les plus adaptés à une exécution sur des machines disposant de ressources limitées ?

La littérature actuelle fournit peu de réponses consolidées à ces interrogations, ce qui justifie la nécessité d'une évaluation comparative rigoureuse des SLM appliqués au problème Text-to-SQL.

1.8.3 Manque d'études sur les architectures hybrides

Les approches Text-to-SQL existantes reposent généralement sur deux paradigmes distincts.

Le premier consiste à utiliser directement un modèle de langage pour générer la requête SQL à partir de la question utilisateur et du schéma de la base de données. Cette approche est simple à mettre en œuvre mais demeure sensible aux hallucinations, aux erreurs de raisonnement et aux limites de contexte.

Le second paradigme repose sur des techniques de récupération d'information, de raisonnement intermédiaire ou de décomposition de tâches afin de guider la génération SQL.

Cependant, relativement peu de travaux explorent de manière approfondie des architectures hybrides combinant simultanément :

- Un mécanisme de récupération contextuelle (RAG) .
- Une représentation enrichie du schéma de données .
- Un raisonnement guidé par des prompts spécialisés .
- Un Small Language Model exécuté localement.

Or, ces architectures hybrides pourraient permettre de compenser les limitations intrinsèques des SLM en leur fournissant des connaissances contextuelles pertinentes sans augmenter la taille du modèle. Cette piste apparaît particulièrement prometteuse dans les environnements où les ressources matérielles sont limitées.

1.8.4 Absence de solutions locales adaptées

La plupart des systèmes Text-to-SQL les plus performants dépendent actuellement de services cloud et d'API commerciales. Cette dépendance soulves plusieurs problématiques importantes :

Confidentialité des données : Dans de nombreux secteurs (administration, santé, finance, industrie), les données manipulées sont sensibles et ne peuvent être transmises à des fournisseurs externes.

Coût d'exploitation : L'utilisation intensive d'API de modèles propriétaires engendre des coûts récurrents qui peuvent devenir prohibitifs pour les petites structures ou les institutions académiques.

Dépendance technologique : Les organisations deviennent dépendantes de fournisseurs tiers pour l'accès aux modèles et aux infrastructures de calcul.

Latence et disponibilité : Les performances du système peuvent être affectées par les contraintes réseau ou les limitations imposées par les fournisseurs de services.

Malgré l'intérêt croissant pour les modèles open source, les solutions Text-to-SQL entièrement locales restent encore peu nombreuses. De plus, les travaux existants se concentrent souvent sur l'amélioration de la précision sans considérer explicitement les contraintes de déploiement réel sur des postes de travail standards.

Cette situation met en évidence le besoin de systèmes capables de fonctionner localement tout en conservant des performances satisfaisantes pour les utilisateurs finaux.

1.8.5 Justification scientifique du projet ASK-DATA

Le projet ASK-DATA s'inscrit précisément dans cette problématique scientifique. Son objectif est de concevoir et d'évaluer un système Text-to-SQL local capable d'interroger des bases de données relationnelles à partir du langage naturel sans recourir à des services cloud externes. La contribution proposée se distingue des travaux existants selon plusieurs axes :

- L'utilisation de Small Language Models open source exécutables localement .
- L'étude comparative de plusieurs SLM récents afin d'identify les modèles les plus adaptés au Text-to-SQL .

- L'intégration d'une architecture hybride combinant compréhension du schéma, enrichissement contextuel et génération SQL .
- L'évaluation simultanée des performances fonctionnelles et des contraintes de déploiement (temps d'inférence, mémoire consommée, taille du modèle) .
- La prise en compte des exigences de confidentialité et de souveraineté des données.

D'un point de vue scientifique, ASK-DATA vise ainsi à répondre à la question de recherche suivante :

Dans quelle mesure un Small Language Model exécuté localement, éventuellement enrichi par des mécanismes contextuels et des techniques d'ingénierie de prompts, peut-il fournir des performances Text-to-SQL comparables à celles des solutions basées sur des Large Language Models tout en réduisant les coûts et en préservant la confidentialité des données ?

1.9 Conclusion

Ce chapitre a présenté les fondements théoriques et l'état de l'art relatifs aux systèmes Text-to-SQL, en mettant en évidence leur rôle dans la démocratisation de l'accès aux bases de données à travers l'utilisation du langage naturel. Après avoir défini le problème **Text-to-SQL** et examiné ses principales applications, nous avons analysé les défis scientifiques associés à cette tâche, notamment la compréhension sémantique des requêtes, le *schema linking*, la génération de requêtes SQL complexes et la généralisation à de nouveaux domaines.

L'étude des approches existantes a montré une évolution progressive des systèmes Text-to-SQL, depuis les méthodes basées sur des règles jusqu'aux architectures neuronales modernes reposant sur les Large Language Models. Les travaux récents démontrent que les LLM ont considérablement amélioré les performances de génération SQL grâce à leurs capacités avancées de compréhension et de raisonnement. Toutefois, ces modèles présentent plusieurs limites importantes liées à leur coût d'utilisation, leurs besoins élevés en ressources matérielles, leur dépendance aux infrastructures cloud et les préoccupations de confidentialité qu'ils soulèvent.

L'analyse critique de la littérature a également permis d'identifier plusieurs lacunes de recherche. En particulier, les évaluations comparatives des Small Language Models dans le contexte Text-to-SQL demeurent limitées, les architectures hybrides exploitant efficacement ces modèles sont encore peu étudiées, et les solutions entièrement locales adaptées aux contraintes de confidentialité et de souveraineté des données restent rares.

Ces constats justifient pleinement l'intérêt du projet **ASK-DATA**, dont l'objectif est de concevoir et d'évaluer un système Text-to-SQL local reposant sur des Small Language Models open source. En proposant une approche capable de fonctionner sans dépendance à des services externes tout en conservant des performances satisfaisantes, ce travail vise à contribuer au développement de solutions Text-to-SQL plus accessibles, économiques et adaptées aux besoins des organisations soucieuses de la protection de leurs données.

Le chapitre suivant sera consacré à la conception et à l'architecture du système **ASK-DATA**, en détaillant les choix technologiques retenus, les modèles étudiés ainsi que les mécanismes mis en œuvre pour assurer la traduction efficace des requêtes en langage naturel vers des requêtes SQL exécutables.

Chapitre 2

Étude des approches SLM pour la génération Text-to-SQL

2.1 Introduction

Après avoir présenté dans le chapitre précédent les fondements théoriques des systèmes **Text-to-SQL** et des **Small Language Models (SLM)**, ce chapitre s'intéresse à l'étude comparative des différentes architectures susceptibles d'être utilisées pour la génération automatique de requêtes **SQL** à partir de questions formulées en langage naturel.

L'objectif principal de cette étude est d'identifier l'approche la plus adaptée au développement d'un système local intelligent capable d'assister les utilisateurs dans l'interrogation de bases de données relationnelles sans nécessiter de connaissances approfondies du langage **SQL**. Contrairement aux solutions reposant sur des **Large Language Models** hébergés dans le cloud, les architectures étudiées doivent répondre à plusieurs contraintes : exécution locale, faible consommation de ressources, préservation de la confidentialité des données et qualité satisfaisante des requêtes **SQL** générées.

Dans cette perspective, quatre approches ont été retenues pour l'évaluation expérimentale :

- l'approche basée sur un **SLM pur**.
- l'approche **SLM augmentée par récupération d'informations (RAG)**.
- l'approche **Multi-Agent**.
- l'approche **PANDAS + SLM (PandasAI)**.

Les expérimentations sont réalisées sur le benchmark **Spider**, largement utilisé dans la littérature pour l'évaluation des systèmes **Text-to-SQL**. Les résultats obtenus permettront d'orienter le choix de l'architecture qui servira de base au développement du système **ASK-DATA (Intelligent Local Web Service)**.

2.2 Approche 1 : SLM pur

2.2.1 Architecture proposée

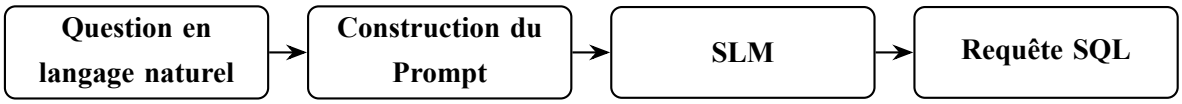
L'approche **SLM pur** constitue l'architecture la plus simple étudiée dans ce mémoire. Elle repose exclusivement sur les capacités internes d'un **Small Language Model** sans mécanisme externe de récupération d'informations ni système de raisonnement distribué.

L'architecture comporte trois composants principaux :

- une question formulée en langage naturel.
- un mécanisme de construction du prompt.
- un **SLM** chargé de générer la requête **SQL** correspondante.

Le modèle reçoit directement une question utilisateur et doit produire une requête **SQL** correcte permettant d'obtenir les informations demandées dans la base de données.

Cette approche permet d'évaluer les capacités intrinsèques du modèle à comprendre l'intention de l'utilisateur et à la traduire en une requête **SQL** valide.



2.2.2 Construction des prompts

La qualité des résultats dépend fortement de la formulation du prompt transmis au modèle. Le prompt doit fournir suffisamment d'informations pour guider le raisonnement du **SLM** tout en restant suffisamment compact pour respecter la fenêtre de contexte disponible.

Une structure typique de prompt peut être la suivante :

> **Générer une requête SQL répondant à la question suivante.**
> **Question :** "Afficher la liste des livres écrits par Charl"
> **Schéma :** Livres(id_livre, titre, auteur)

Des versions plus élaborées peuvent inclure :

- le schéma complet de la base de données.
- la description des colonnes.
- les relations entre les tables.

- des exemples de questions et de requêtes **SQL** correspondantes.
- des contraintes de génération.

L'objectif est de réduire les ambiguïtés et d'améliorer l'exactitude des requêtes **SQL** générées.

2.2.3 Processus d'inférence

Le processus d'inférence comprend les étapes suivantes :

- **Réception :** accueil de la question formulée par l'utilisateur.
- **Génération du prompt :** construction du contexte et intégration des instructions.
- **Exécution du modèle :** traitement du prompt par le **SLM**.
- **Génération de la requête :** production de la requête **SQL** finale.

Cette architecture présente plusieurs avantages :**simplicité** de mise en œuvre,**faible coût** computationnel **facilité** de déploiement local.

Limites du système :

Cependant, les performances peuvent diminuer lorsque les questions deviennent complexes ou lorsque des connaissances contextuelles supplémentaires (telles que des règles métiers ou des jointures multiples) sont nécessaires.

2.3 Approche 2 : SLM + RAG

2.3.1 Principes du RAG

Le Retrieval-Augmented Generation (**RAG**) combine les capacités génératives d'un modèle de langage avec un mécanisme de récupération d'informations externes. L'idée fondamentale est d'enrichir le contexte fourni au modèle avant la génération de la réponse.

Dans le cadre du **Text-to-SQL**, cette approche permet de compléter la question de l'utilisateur par des informations spécifiques relatives aux tables, aux colonnes, aux relations entre entités, aux règles métier et aux métadonnées de la base. Le modèle dispose ainsi d'un contexte enrichi pour générer une requête **SQL** exacte, robuste et parfaitement adaptée au schéma de la base de données.(Nouali et al., 2025).

2.3.2 Construction de la base documentaire

La base documentaire constitue le cœur du système RAG. Elle peut être construite à partir des éléments suivants : du schéma relationnel, du dictionnaire de données, des descriptions métier, d'exemples de questions et de requêtes SQL annotées, de documents techniques.

Chaque document est converti en représentations vectorielles permettant une recherche sémantique efficace. L'objectif est de fournir au système des connaissances spécifiques à la base de données étudiée.

2.3.3 Processus de récupération d'informations

Lorsqu'une question en langage naturel est soumise, les étapes suivantes sont exécutées :

1. La question est analysée.
2. Un vecteur de représentation est calculé.
3. Les documents les plus pertinents sont recherchés.
4. Les informations récupérées sont ajoutées au prompt.

Cette étape permet d'apporter un contexte supplémentaire avant la génération de la requête SQL.

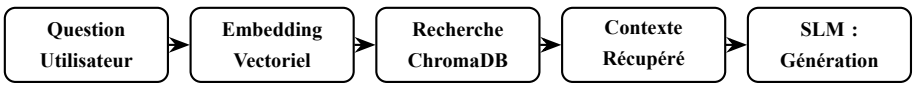


FIG. 2.3 : Processus de récupération d'informations (RAG).

2.3.4 Génération augmentée

Le prompt final est constitué des éléments suivants : la question utilisateur, les documents récupérés et les instructions de génération. Le SLM utilise alors ces informations pour produire une requête SQL plus précise. Cette approche permet généralement une **meilleure compréhension du schéma**, une **réduction des ambiguïtés** ainsi qu'une **amélioration de la qualité des requêtes SQL générées**.

2.4 Approche 3 : Architecture Multi-Agent

2.4.1 Concepts des systèmes multi-agents

Les systèmes multi-agents reposent sur la coopération de plusieurs agents spécialisés travaillant ensemble pour résoudre une tâche complexe.

Contrairement à une architecture monolithique, chaque agent possède une responsabilité précise.

Cette décomposition favorise : la spécialisation, la modularité, la qualité du raisonnement.

2.4.2 Décomposition fonctionnelle

L'architecture proposée est composée de trois agents principaux :

Agent d'analyse de l'intention, agent générateur SQL, agent évaluateur.

FIG. 2.4 : Architecture multi-agents pour la génération SQL.

2.4.3 Agent d'analyse et de contextualisation

Cet agent assure l'analyse de la question utilisateur.

Ses principales missions sont :

- identifier les entités mentionnées.
- détecter les attributs recherchés.
- reconnaître les conditions de filtrage.
- identifier les opérations d'agrégation.
- déterminer les relations entre les tables concernées.

L'agent produit une représentation structurée de l'intention de l'utilisateur qui sera utilisée par les agents suivants.

2.4.4 Agent générateur SQL

L'agent générateur reçoit :

- la représentation de l'intention utilisateur.
- les informations relatives au schéma.
- le contexte métier disponible.

Il transforme ces informations en une requête SQL conforme au schéma de la base de données.

Son rôle est similaire à celui d'un développeur chargé de traduire un besoin métier en requête SQL exécutable.

2.4.5 Agent évaluateur

L'agent évaluateur agit comme un mécanisme de contrôle qualité.
Ses principales missions sont :

- vérifier la validité syntaxique de la requête.
- vérifier sa cohérence sémantique.
- détecter les tables ou colonnes manquantes.
- corriger les ambiguïtés.
- optimiser la requête produite.

Cette étape permet d'obtenir des requêtes SQL généralement plus fiables que celles générées par un seul agent.

2.5 Approche 4 : PANDAS + SLM (PandasAI)

2.5.1 Principe général

Contrairement aux approches Text-to-SQL classiques qui génèrent directement une requête SQL, l'approche PANDAS + SLM, inspirée du framework PandasAI, adopte une stratégie différente basée sur la génération automatique de code Python.

Dans cette architecture, le modèle reçoit une question formulée en langage naturel et génère un script Python permettant de manipuler directement les données stockées dans un objet Pandas DataFrame.

Cette approche peut être considérée comme un système de type Text-to-Python-Code dans lequel le modèle traduit l'intention de l'utilisateur en instructions Python exécutables.

L'objectif est d'exploiter les capacités de raisonnement et de génération des Small Language Models afin d'automatiser l'analyse des données sans exiger de l'utilisateur une connaissance préalable du langage SQL ou des bibliothèques Python.

2.6 Conclusion

Ce chapitre a présenté quatre approches basées sur les Small Language Models pour la génération de requêtes SQL à partir de questions formulées en langage naturel. L'étude a débuté par l'approche **SLM pur**, qui constitue la solution la plus simple et la moins coûteuse en ressources. Elle a ensuite été enrichie par l'intégration d'un mécanisme de **Retrieval-Augmented Generation (RAG)** permettant d'améliorer la compréhension des questions et du schéma de la base grâce à l'apport d'informations contextuelles externes.

Une architecture **Multi-Agent** a également été étudiée afin de décomposer le processus de génération SQL en plusieurs tâches spécialisées, favorisant ainsi un raisonnement plus structuré et une meilleure qualité des requêtes produites.

Enfin, l'approche **PANDAS + SLM (PandasAI)** a été analysée comme une alternative aux systèmes Text-to-SQL traditionnels. Cette solution repose sur la génération automatique de code Python permettant d'interroger et d'analyser les données sans produire explicitement de requêtes SQL. Elle s'inscrit ainsi dans une logique de type **Text-to-Code**, où le langage naturel est converti en instructions exécutables sur un DataFrame Pandas.

Cette étude théorique constitue le cadre méthodologique des expérimentations présentées dans le chapitre suivant. Les évaluations réalisées sur le benchmark **Spider** permettront d'identifier l'approche offrant le meilleur compromis entre précision des requêtes SQL générées, coût computationnel, simplicité de déploiement local et adéquation aux contraintes du système intelligent ASK-DATA.

Chapitre 3

Méthodologie expérimentale et protocole d'évaluation

3.1 Introduction

Ce chapitre décrit l'environnement expérimental mis en place pour évaluer les différentes approches SLM présentées au chapitre précédent. Nous présentons successivement les configurations matérielles et logicielles utilisées, le jeu de données expérimental (benchmark Spider), les dix modèles SLM étudiés, les protocoles d'expérimentation retenus pour chaque phase (sélection des SLM puis évaluation des architectures avancées), les métriques d'évaluation et les conditions de validité expérimentale.

L'ensemble des expérimentations a été réalisé entièrement en local, sans recours à aucun service cloud, ce qui répond à la contrainte fondamentale de confidentialité du projet ASK-DATA. Le logiciel **LM Studio** a été utilisé comme interface principale pour le chargement et l'exécution des modèles.

Notre démarche expérimentale s'articule autour de **deux phases distinctes et complémentaires** :

- **Phase 1 — Sélection des SLM :** Nous comparons dix modèles SLM selon l'approche SLM simple (Text-to-SQL pur), en les évaluant sur deux environnements matériels distincts (CPU et GPU). Cette phase a pour objectif d'identifier les trois modèles les plus performants qui serviront de base aux expérimentations de la phase suivante.
- **Phase 2 — Évaluation des architectures avancées :** Les trois meilleurs SLM retenus sont ensuite intégrés dans quatre architectures avancées : SLM + RAG, Multi-Agent, PandasAI et génération en langue arabe. Cette phase permet d'analyser l'apport de chaque architecture par rapport à l'approche de référence.

Cette structuration en deux phases garantit une démarche scientifiquement rigoureuse : les choix architecturaux sont fondés sur des résultats objectifs de sélection, et les comparaisons entre architectures s'appuient sur un ensemble de modèles homogène et contrôlé.

3.2 Environnement expérimental

3.2.1 Configuration matérielle

Deux plateformes matérielles distinctes ont été utilisées pour évaluer l'influence du hardware sur les performances des SLM.

Plateforme CPU :

TAB. 3.1 : Configuration technique du PC

Composant	Spécification
Processeur	AMD Ryzen 5 3550H (2.10 GHz)
RAM	16,0 Go
Stockage	238 Go SSD + 932 Go HDD
Carte graphique	NVIDIA GeForce GTX 1650 (4 Go)
Système d'exploitation	Windows 10 Famille (22H2)

Plateforme GPU :

TAB. 3.2 : Configuration de la plateforme GPU

Composant	Spécification
GPU	NVIDIA GeForce GTX 1650
VRAM	4 Go GDDR5
Pilote CUDA	CUDA 12.8

3.2.2 Configuration logicielle

TAB. 3.3 : Versions des outils utilisés par environnement

Logo	Outil	Environnement1	Environnement2
	LM Studio	0.4.12	0.4.12
	Python	3.11.15	3.11.14
	Streamlit	1.57.0	1.54.0
	SQLite	3.51.2	3.51.1
	ChromaDB	1.5.9	aucun
	Pandas	3.0.3	2.3.3
	OpenAI	2.37.0	1.109.1

3.2.3 Frameworks et bibliothèques utilisés

L'architecture expérimentale repose sur les composants logiciels suivants :

- **LM Studio** : Interface graphique permettant de charger, quantifier et exposer des modèles SLM via une API REST compatible OpenAI, fonctionnant entièrement en local. LM Studio prend en charge les formats GGUF et GGML, qui sont les formats de quantification les plus répandus pour les SLM locaux.
- **SQLite** : Moteur de base de données léger utilisé pour stocker les bases de données du benchmark Spider Yu et al. (2018) et exécuter les requêtes générées.
- **ChromaDB** Chroma (2023) : Base de données vectorielle locale dédiée à l'implémentation du module RAG.
- **Pandas** : Bibliothèque Python centrale pour la manipulation et l'analyse des jeux de données tabulaires, utilisée aussi pour l'approche Pandas + SLM.
- **OpenAI** : Bibliothèque client utilisée pour gérer les communications entre le code et l'API locale.
- **Streamlit** : Framework Python utilisé pour créer l'interface web interactive de l'application.

3.3 Jeu de données expérimental : Spider

3.3.1 Présentation du dataset Spider

Le benchmark **Spider** constitue actuellement le principal standard d'évaluation des systèmes Text-to-SQL *cross-domain* dans la littérature scientifique Yu et al. (2018). Il a été développé par l'Université de Yale et publié en 2018. Il est composé de :

- **10 181** paires (question en langage naturel, requête SQL annotée) dont **5 693** requêtes SQL uniques.
- **200** bases de données couvrant **138** domaines différents (vols, musique, sport, administration, etc.).
- des requêtes de complexité variable (simple, modérée, difficile, extra-difficile).

Spider est *cross-domain* : les bases de données de l'ensemble de test sont différentes de celles utilisées pour l'entraînement, ce qui évalue la capacité de généralisation des modèles.

3.3.2 Structure des bases de données

Les bases de données Spider sont des bases **SQLite** multi-tables. Chaque base est accompagnée :

- d'un fichier `tables.json` décrivant le schéma (tables, colonnes, clés primaires, clés étrangères).
- de fichiers `.sqlite` contenant les données réelles.

La richesse des schémas varie considérablement, allant de bases à 2 tables jusqu'à des schémas comportant plus de 10 tables interconnectées.

3.3.3 Répartition des données

Pour nos expérimentations, nous avons sélectionné un sous-ensemble de **400 questions** par échantillonnage stratifié afin de préserver une représentation équilibrée des différents niveaux de complexité SQL présents dans Spider. Ces questions sont réparties selon cinq catégories de complexité. Ces catégories, définies par la structure syntaxique et les clauses SQL requises, constituent une **taxonomie interne** proposée pour faciliter l'analyse des performances selon les structures SQL, et ne remplace pas la classification officielle du benchmark Spider (*Easy*, *Medium*, *Hard*, *Extra Hard*) :

Tab. 3.4 : Répartition des 400 questions sélectionnées et critères syntaxiques associés.

Catégorie	Critères SQL	Nombre	Pourcentage
Simple	`COUNT`, `SELECT`	80	20%
Filtrage	`WHERE`, `BETWEEN`	80	20%
Agrégation	`GROUP BY`, `HAVING`	80	20%
Complexe	`JOIN`, Sous-requêtes	90	22,5%
Ambiguë	`ORDER BY`, `LIMIT`	70	17,5%
Total	-	**400**	**100%**

Ce même ensemble de 400 questions a été utilisé pour toutes les expérimentations, garantissant ainsi la comparabilité des résultats.

3.3.4 Prétraitement et préparation des requêtes

Avant les expérimentations, les données Spider ont été soumises à un prétraitement :

1. **Extraction des schémas** : Pour chaque question, le schéma de la base de données correspondante est extrait et formaté pour être inclus dans le prompt.

2. **Normalisation des requêtes de référence** : Les requêtes SQL de référence (*gold queries*) sont normalisées (mise en minuscules des mots-clés, suppression des espaces superflus) pour faciliter la comparaison avec les requêtes générées.

3. **Préparation de la version arabe** : Nous avons traduit les 400 questions de l'anglais vers l'arabe à l'aide d'un script Python. Par la suite, nous avons utilisé **Claude Sonnet (Anthropic, version de juin 2026)** Anthropic (2026) pour valider et confirmer la qualité et la fidélité de chaque traduction.

3.4 Small Language Models étudiés

3.4.1 Critères de sélection des SLM

Les dix modèles SLM ont été sélectionnés selon les critères suivants :

- **Compatibilité locale** : Disponibilité au format GGUF compatible avec LM Studio pour un déploiement local.
- **Taille** : Moins de 10 milliards de paramètres, permettant une exécution fluide sur CPU grand public.
- **Diversité architecturale** : Représentation variée de familles de modèles (Deepseek, Mistral, Phi, Qwen, etc.).
- **Spécialisation** : Inclusion équilibrée de deux types de modèles :
  - **Modèles généralistes** : Modèles entraînés sur des données textuelles variées pour évaluer leurs capacités de raisonnement logique de base.
  - **Modèles spécialisés** : Modèles spécifiquement pré-entraînés ou affinés (*fine-tunés*) sur du code et du SQL pour maximiser la précision syntaxique.
- **Performance** : Préférence pour les modèles démontrant une aptitude reconnue dans la compréhension et la génération de requêtes SQL.

3.4.2 Présentation des dix modèles évalués

Dans le cadre de nos expérimentations, nous avons sélectionné dix modèles de langage (SLM) de tailles variées. Cette sélection vise à couvrir un large spectre d'architectures, allant de modèles généralistes de haute performance à des modèles hautement spécialisés dans le domaine du code et du langage SQL. Cette diversité nous permet d'évaluer l'impact de l'architecture et du domaine d'entraînement sur la précision des requêtes générées en environnement local.

3.4.3 Caractéristiques techniques des modèles

Le tableau 3.5 synthétise les spécifications techniques des dix modèles SLM sélectionnés pour nos expérimentations. Cette sélection repose sur un équilibre entre capacité de raisonnement, besoins en ressources matérielles et compatibilité avec un déploiement local via le format GGUF. Les modèles retenus couvrent trois catégories distinctes — *Spécialisé SQL*, *Généraliste* et *Edge* — afin de garantir une évaluation comparative large et représentative des approches SLM disponibles en open-source. Chaque modèle a été chargé via **LM Studio** dans sa version quantifiée GGUF, assurant ainsi une compatibilité totale avec les contraintes de déploiement local imposées par le projet ASK-DATA.

TAB. 3.5 : Caractéristiques techniques des dix SLM évalués.

N°	Modèle	Entreprise	Catégorie	Params	Quant.	Contexte	Taille
1	XiYanSQL-QwenCoder-7B	XGenerationLab	Spécialisé SQL	7B	Q4_K_S	64k	4.5 Go
2	SLM-SQL-1.5B	cycloneboy	Spécialisé SQL	1.5B	Q8_0	32k	1.9 Go
3	SQLCoder-7B-2	Defog AI	Spécialisé SQL	7B	Q5_K_M	16k	4.8 Go
4	Gemma2B Text-to-SQL Expert	Google (ukung)	Spécialisé SQL	2B	Q6_K	8k	2.1 Go
5	Qwen3.5-4B	Alibaba Cloud	Généraliste	4B	Q4_K_M	128k	3.4 Go
6	Qwen3.5-9B	Alibaba Cloud	Généraliste	9B	Q4_K_M	128k	6.5 Go
7	DeepSeek-R1-Distill-Qwen-7B	DeepSeek	Généraliste	7B	Q4_K_M	128k	4.7 Go
8	DeepSeek-R1-Distill-Qwen-1.5B	DeepSeek	Edge	1.5B	Q8_0	128k	1.9 Go
9	Ministral 3B Instruct	Mistral AI	Edge	3B	Q4_K_M	128k	3.0 Go
10	IBM Granite 4.0 Tiny	IBM	Edge	1.4B	Q4_K_M	128k	4.2 Go

Analyse des caractéristiques.

- **Nombre de paramètres** : Les modèles varient de 1,4B à 9B paramètres, offrant un spectre allant des modèles Edge ultra-compacts aux modèles généralistes à plus grande capacité de raisonnement.
- **Entreprise** : La sélection couvre des acteurs spécialisés SQL (XGenerationLab, Defog AI), des laboratoires généralistes (Alibaba Cloud, DeepSeek, Mistral AI) et des entreprises infrastructure (IBM), permettant d'évaluer l'impact de l'origine du modèle sur ses performances.
- **Catégorie** : Les modèles sont répartis en trois familles — *Spécialisé SQL* (fine-tunés sur Spider/BIRD), *Généraliste* (multilingues et multi-tâches) et *Edge* (conçus pour ressources limitées) — couvrant ainsi différentes philosophies de conception.
- **Quantification** : Tous les modèles utilisent le format GGUF avec des niveaux allant de Q4_K_S à Q8_0, offrant le meilleur compromis entre vitesse d'inférence, occupation

mémoire (1,9 Go à 6,5 Go) et qualité des réponses.

- **Fenêtre de contexte** : Elle varie de 8k à 128k tokens, paramètre critique pour l'architecture RAG car elle détermine la quantité de schémas et métadonnées injectables dans le prompt.
- **Taille mémoire** : Entre 1,9 Go et 6,5 Go selon la quantification appliquée, contrainte directement liée aux exigences de déploiement local du projet ASK-DATA.

3.5 Protocole expérimental de sélection des SLM

3.5.1 Approche de référence : Text-to-SQL avec SLM pur

La première phase de l'expérimentation consiste à évaluer les dix modèles SLM sur l'approche SLM pur avec les 400 questions Spider. Cette phase a pour but unique de **sélectionner les trois meilleurs modèles** qui seront ensuite utilisés pour l'évaluation des architectures avancées (RAG, Multi-Agent, Pandas, Arabe).

3.5.2 Paramétrage des expériences et configuration matérielle

Paramètres de génération :

Tous les modèles ont été testés avec les mêmes paramètres de génération afin d'assurer la cohérence et la reproductibilité des résultats. Ces paramètres contrôlent le comportement du décodage du modèle lors de la génération de la requête SQL et ont été fixés après une phase de calibration préliminaire.

TAB. 3.6 : Paramètres de génération et justifications techniques.

Paramètre	Valeur	Description et justification
Temperature	0.0	Comportement déterministe : le modèle choisit toujours le token de plus haute probabilité, garantissant la reproductibilité des expériences.
Top-P	0.1	Restreint la sélection aux tokens les plus probables (10% cumulé), réduisant le risque de générer une syntaxe SQL invalide.
Top-K	40	Limite le vocabulaire candidat aux 40 meilleurs tokens, réduisant l'espace de recherche et stabilisant la génération.
Repeat Penalty	1.1	Pénalise les répétitions de tokens, prévenant les bégaiements syntaxiques sur les clauses SQL (`WHERE`, `AND`).
Max Tokens	2048	Couvre les requêtes SQL complexes (jointures, sous-requêtes) tout en évitant les générations infinies.
Stop Tokens	`[;, ```]`	Arrêt immédiat après le ; final ou les backticks, assurant une sortie propre et directement exécutable.

Prompt système et construction du contexte :

La qualité de la génération SQL dépend fortement de la formulation du prompt transmis au modèle. Nous avons adopté une architecture en deux parties : un **system prompt** définissant le rôle et les contraintes du modèle, et un **user prompt** fournissant le schéma de la base de données et la question en langage naturel.

System prompt : Le system prompt suivant a été utilisé de manière uniforme pour tous les modèles :

System Prompt — Génération SQL

```
SYSTEM_PROMPT = """

You are an expert SQL query generator.
Given a natural language question and a database
schema, generate ONLY the SQL query.
No explanation, no markdown, no preamble.

IMPORTANT: Use ONLY SQLite-compatible syntax and functions.
Do NOT use MySQL, PostgreSQL, SQL Server or Oracle functions.

Allowed: strftime(), COALESCE(), CAST(), SUBSTR(),
LENGTH(), LOWER(), UPPER(), ROUND().

Forbidden: DATE_FORMAT(), IFNULL(), TOP, ROWNUM,
RIGHT JOIN, FULL OUTER JOIN."""
```

Ce prompt impose trois contraintes fondamentales au modèle :

- **Rôle expert** : Le prompt spécialise explicitement le modèle dans la tâche de génération Text-to-SQL et réduit l'espace de sortie vers des séquences syntaxiquement structurées.
- **Sortie exclusive** : l'instruction *« generate ONLY the SQL query »* supprime toute tendance du modèle à produire des explications ou du texte introductif, rendant la sortie directement parsable.
- **Absence de formatage** : l'interdiction explicite du Markdown et des préambules garantit une sortie brute compatible avec le pipeline d'exécution automatique.
- **Contrainte SQLite** : Les bases de données Spider étant au format **SQLite**, le prompt interdit explicitement les fonctions propres à d'autres SGBD (`DATE_FORMAT()`, `TOP`, `ROWNUM`) et impose l'usage des fonctions SQLite natives (`strftime()`, `COALESCE()`, `CAST()`, `ROUND()`), afin d'éviter les erreurs d'exécution liées aux incompatibilités de dialecte SQL, et améliore le taux d'*Execution Accuracy* mesuré lors de l'évaluation.

Stratégie d'exécution matérielle :

Test 1 : Exécution sur CPU (Baseline) : Dans cette configuration, les modèles sont exécutés exclusivement sur le processeur (CPU) et la mémoire vive (RAM) du système. Ce mode sert de référence pour évaluer la capacité d'inférence intrinsèque des modèles sans aucune accélération matérielle.

Test 2 : Exécution avec accélération GPU (NVIDIA GTX 1650 - 4 Go) : Pour l'accélération GPU, nous avons divisé les modèles en deux groupes selon leur taille et les capacités de notre VRAM :

TAB. 3.7 : Modes d'exécution sur GPU.

Mode	Groupe	Configuration
Full GPU	< 3B params	100% des couches sur GPU
Hybride	7B à 9B params	16 à 24 couches sur GPU (reliquat CPU)

Justification des choix matériels (GPU).

- **Mode "Full GPU"** : Les modèles légers (ex : SLM-SQL-1.5B, Granite Tiny) sont assez compacts pour tenir entièrement dans les 4 Go de VRAM. Cela permet une inférence 100% GPU, garantissant une vitesse maximale sans solliciter le CPU.
- **Mode "Hybride"** : Les modèles plus volumineux (ex : Qwen 9B, SQLCoder 7B) dépassent la capacité de la VRAM. En saturant la carte graphique (environ 3,5 Go utilisés) avec 16 à 24 couches, nous traitons le reste via le CPU. Cette méthode permet d'exécuter des modèles normalement trop lourds tout en améliorant significativement le temps de réponse par rapport à un traitement purement CPU, tout en évitant les instabilités système.

Cette double configuration CPU/GPU permet une évaluation complète et équilibrée : **le mode CPU** établit une référence de base accessible sur tout matériel standard, tandis que **les modes Full GPU et Hybride** exploitent au maximum les 4 Go de VRAM disponibles selon la taille du modèle. Chaque modèle est évalué dans la meilleure configuration compatible avec les contraintes matérielles disponibles, rendant les comparaisons de performance scientifiquement significatives dans le contexte d'un déploiement local contraint.

3.5.3 Protocole d'évaluation CPU et GPU

Chaque modèle a été chargé dans LM Studio successivement sur les deux environnements matériels (CPU puis GPU) pour traiter les 400 questions du benchmark. La procédure expérimentale pour chaque question suit les étapes suivantes :

1. **Construction du prompt** : Le schéma de la base de données, la question en langage naturel et les instructions de génération sont concaténés.
2. **Génération SQL** : Le modèle produit la requête SQL.
3. **Mesure temporelle** : Nous enregistrons deux indicateurs de performance :
    - *Temps de génération SQL seul* : Le temps nécessaire au modèle pour produire uniquement la requête SQL.
    - *Temps total* : Le temps global incluant la génération et l'affichage complet de la réponse.
4. **Exécution** : La requête SQL générée est exécutée sur SQLite afin d'évaluer le résultat (Execution Accuracy).

Pour l'environnement GPU, LM Studio offre la possibilité de décharger une partie ou la totalité des couches du modèle sur le GPU (*GPU offloading*). Pour chaque modèle, le nombre de couches déchargées a été maximisé en fonction de la VRAM disponible, selon la stratégie Full GPU ou Hybride décrite précédemment.

3.5.4 Critères de sélection des meilleurs SLM

La sélection des trois meilleurs modèles repose sur un score composite équilibrant la performance analytique et l'efficacité matérielle. Ce score agrège les indicateurs suivants :

- **Précision (EM et EX)** : L'*Exact Match* (EM) et l'*Execution Accuracy* (EX) obtenus sur les deux environnements d'accélération (CPU et GPU).
- **Qualité qualitative** : Le score attribué par l'approche *LLM-as-Judge* sur les requêtes générées.
- **Efficacité temporelle** : Le temps d'inférence moyen mesuré dans deux contextes distincts :
    - Temps de réponse en mode **CPU seul** .
    - Temps de réponse en mode **GPU** (Full ou Hybride).

3.6 Protocoles d'évaluation des architectures avancées

Les trois meilleurs SLM sélectionnés à l'issue de la phase précédente ont été utilisés pour évaluer les quatre architectures avancées sur les mêmes 400 questions Spider.

3.6.1 Architecture SLM + RAG

Pour l'expérience RAG Lewis et al. (2020), nous avons adopté la méthodologie suivante :

1. **Indexation** : Les schémas DDL de toutes les bases de données du benchmark Spider ont été indexés dans ChromaDB en utilisant le modèle d'embedding local **sentence-transformers/all-MiniLM-L6-v2**. Pour chaque table, le document indexé contient les colonnes, types, clés primaires, clés étrangères, ainsi que des hints sémantiques générés automatiquement (ex. `SELECT COUNT(*) FROM table`). Les données exemples ne sont **pas** stockées dans ChromaDB — elles sont lues directement depuis SQLite au moment du retrieval, gardant ainsi l'index léger et rapide.

2. **Récupération (Retrieval)** : Pour chaque question, les tables les plus pertinentes sont récupérées via une recherche de similarité sémantique, puis injectées dynamiquement dans le prompt. Avant de fixer la configuration définitive, nous avons conduit une phase de **sélection des hyperparamètres RAG** portant sur trois dimensions :

   - **Métrique de similarité** : Trois métriques ont été comparées — `cosine` (similarité cosinus), `ip` (produit interne) et `l2` (distance euclidienne) — afin d'identifier celle qui produit le meilleur alignement sémantique entre la question et les schémas indexés.
   - **Seuil de filtrage (threshold)** : Pour chaque métrique, plusieurs seuils ont été testés afin d'éliminer les résultats de faible pertinence. Les valeurs testées sont $\{0.10, 0.20, 0.30, 0.40\}$ pour les métriques `cosine` et `ip`, et $\{0.50, 0.80, 1.00, 1.20\}$ pour `l2` (distance croissante).

     Note sur le choix des seuils pour la distance L2 : Contrairement à la similarité cosinus et au produit scalaire (Inner Product), qui mesurent un degré de similarité normalisé entre vecteurs, la distance euclidienne (L2) est une mesure de dissimilarité géométrique dans l'espace vectoriel.
     Elle n'est pas bornée et dépend fortement de la normalisation des embeddings, ce qui entraîne des plages de valeurs différentes par rapport aux mesures de similarité. Ainsi, les seuils utilisés pour L2 (0.5, 0.8, 1.0 et 1.2) ont été ajustés empiriquement afin de correspondre à des niveaux de pertinence comparables à ceux obtenus avec les métriques Cosine et Inner Product.

   - **Nombre de tables récupérées (K)** : Les valeurs $K \in \{1, 2, 3, 5\}$ ont été évaluées expérimentalement. La valeur $K = 3$ a été retenue comme compromis optimal entre richesse contextuelle et taille du prompt.

Chaque combinaison (métrique × seuil) a été évaluée selon deux critères : le taux d'*Execution Accuracy* (EX) et l'*Exact Match* (EM). La configuration optimale retenue est **metric=ip** (produit interne) avec **threshold=0.1**, offrant le meilleur EX sur les modèles testés.

3. **Génération** : Le pipeline RAG envoie au LLM deux prompts complémentaires. Le **system prompt** définit le rôle et les règles de génération, tandis que le **user prompt** est construit dynamiquement pour chaque question avec les schémas récupérés et les données exemples réelles :

System Prompt — RAG Pipeline

```
You are an expert SQL query generator.
Given a question, schema, and sample data,
generate ONLY the SQL query that answers the question.
Rules:
- Output ONLY the SQL query.
- No explanation, no markdown, no backticks.
- Use exact table/column names from schema.
- Use JOINs based on Foreign Keys.
```

User Prompt — RAG Pipeline

```
### Available tables: table1, table2, table3
### Schema with sample data:
Table: table1 | Columns: col1, col2, col3
Types: col1 (INT), col2 (TEXT)
FKs: [col3] → other_table.[id]
Sample: col1 | col2 | col3 | ...
        val1 | val2 | val3 | ...
### Question: <question en langage naturel>
```

4. **Validation et exécution** : La requête SQL générée est validée syntaxiquement via `sqlglot` (Mao, 2023) (vérification des mots-clés interdits, parsing SQLite), puis exécutée sur la base SQLite via `SQLAlchemy`. Le résultat est comparé à la requête de référence pour calculer l'EX et l'EM.

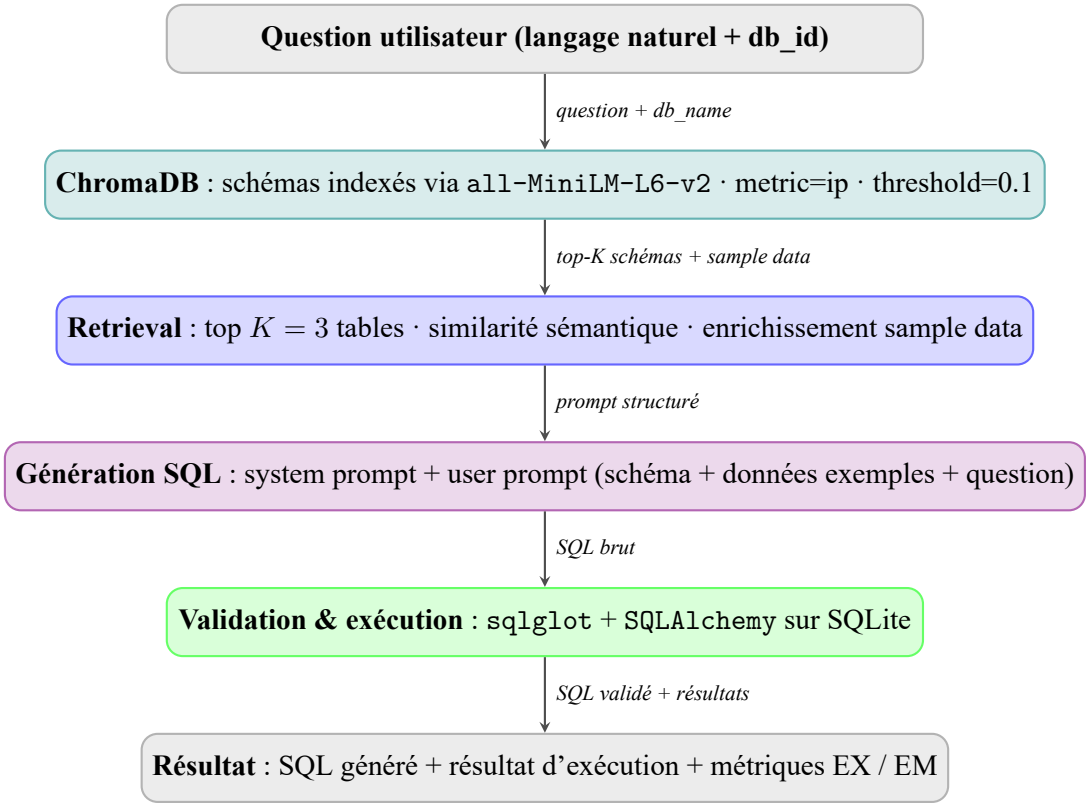


FIG. 3.1 : Pipeline SLM + RAG pour la génération SQL dans ASK-DATA.

3.6.2 Architecture Multi-Agent

Pour l'expérience Multi-Agent, nous avons implémenté un pipeline séquentiel de **quatre agents spécialisés**, chacun ayant un rôle précis dans la chaîne de génération SQL :

1. **Orchestrateur** : Coordonne l'ensemble des agents de manière séquentielle. Il reçoit la question utilisateur et le chemin de la base SQLite, appelle chaque agent dans l'ordre, transmet les résultats d'un agent à l'autre, et retourne le résultat final à l'utilisateur.

2. **Agent 1 — Schema Extractor** : Lit directement la base SQLite, et extrait pour chaque table : la liste des colonnes, les types, les clés primaires, les clés étrangères, ainsi que des données exemples réelles (*sample rows*). Le schéma formaté est ensuite injecté dans le prompt du générateur.

3. **Agent 2 — SQL Generator** : Reçoit le schéma extrait et la question en langage naturel, puis génère une requête SQL via le LLM. Le prompt est structuré en deux parties : un **system prompt** définissant le rôle et les règles, et un **user prompt** construit dynamiquement à chaque question.

System Prompt — Agent 2 (SQL Generator)

```
You are an expert SQL query generator.
Given a question, schema, and sample data,
generate ONLY the SQL query that answers the question.

Rules:
- Output ONLY the SQL query.
- No explanation, no markdown, no backticks.
- Use exact table/column names from schema.
- Use sample data to understand formats.
- Use JOINs based on Foreign Keys.
- End the query with a semicolon.
```

User Prompt — Agent 2 (construit à chaque question)

```
### Available tables: table1, table2, ...

### Schema with sample data:
  Table: table1 | Columns: col1, col2 ...
  Column details: col1 (INTEGER) [PK], col2 (TEXT)
  Foreign Keys: [col2] → other_table.[id]
  Sample data: val1 | val2 | ...

### Hint: This question requires COUNT /
GROUP BY / JOIN (selon opération détectée).

### Question:
<question en langage naturel>
```

4. **Agent 3 — Verifier & Regenerator** : Valide le SQL généré en deux étapes — validation syntaxique via `sqlglot` puis exécution réelle sur SQLite en mode lecture seule. En cas d'erreur, un prompt de régénération intelligent est construit, incluant l'erreur exacte, la liste stricte des colonnes disponibles et les données exemples. Trois tentatives de régénération ont été retenues afin de limiter l'augmentation du temps d'inférence tout en permettant au système de corriger la majorité des erreurs syntaxiques observées durant la phase pilote. Le prompt de régénération utilisé est le suivant :

Prompt — Agent 3 (Verifier & Regenerator)

```
PREVIOUS ATTEMPT FAILED: <erreur>
TABLES: Table1(col1, col2), Table2(colA, colB)
SCHEMA: <schéma et données...>
RULES: Use exact names, prefix columns, output ONLY SQL.
Question: <question en langage naturel>
SQL Query:
```

5. **Agent 4 — Executor** : Reçoit la requête SQL validée, vérifie la sécurité (bloque les opérations d'écriture : `DROP`, `DELETE`, `INSERT`, `UPDATE`), exécute la requête sur SQLite via `SQLAlchemy`, et retourne les résultats sous forme de DataFrame pandas accompagné d'une réponse en langage naturel.

Les agents sont appelés **séquentiellement**, chaque agent recevant en entrée le résultat de l'agent précédent. Le temps total d'inférence inclut les appels à l'ensemble des agents. Les 400 questions du benchmark ont été traitées par chacun des trois modèles SLM sélectionnés afin d'évaluer l'apport du pipeline multi-agent par rapport à l'approche SLM simple.

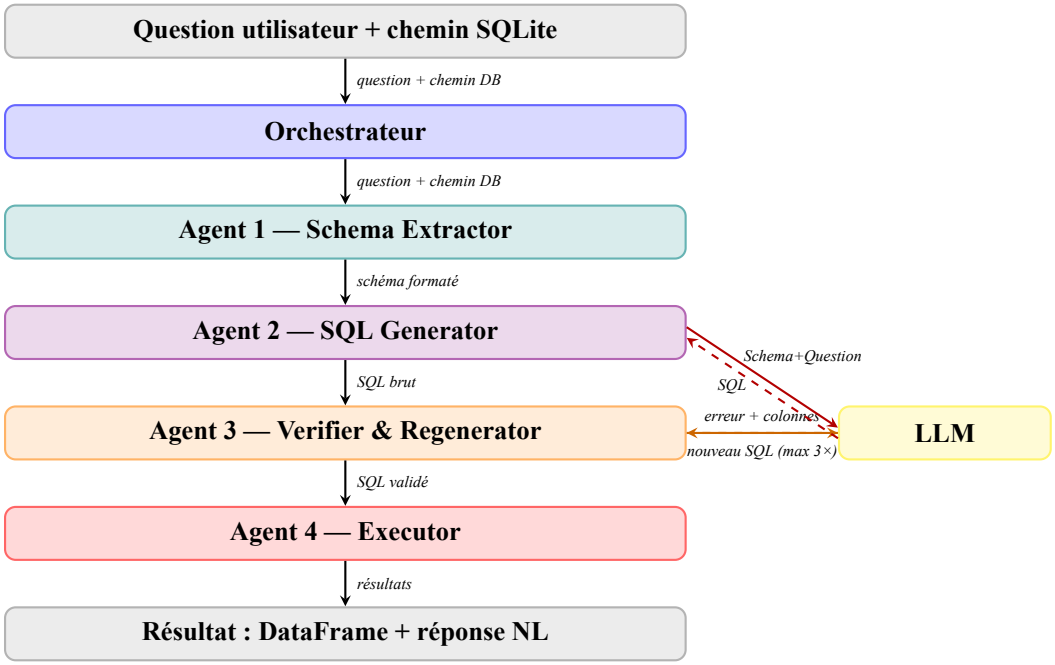


FIG. 3.2 : Pipeline multi-agent pour la génération SQL dans ASK-DATA.

Note

On a changé la température à `0.1` dans cette architecture car l'Agent 3 (Verifier) détecte les erreurs SQL et régénère une nouvelle requête via le LLM. Avec `0.0`, le LLM risque de reproduire exactement le même erreur à chaque tentative. Avec `0.1`, une légère variation permet au LLM de trouver une correction différente.

3.6.3 Architecture Pandas + SLM (PandasAI)

Pour l'expérience Pandas + SLM, nous avons utilisé l'agent **PandasAI** (Galli, 2023) afin de coupler les capacités de raisonnement du SLM avec la puissance de manipulation de données de la bibliothèque Pandas :

1. **Chargement dynamique** : Les bases de données Spider sont chargées depuis SQLite et converties en objets `DataFrame` pandas, puis encapsulées dans des `SmartDataframe` PandasAI. Chaque table est indexée avec son nom, ses colonnes et ses données exemples, permettant au SLM de sélectionner dynamiquement le bon `dfs[i]` selon la question posée.

2. **Ingénierie de prompt** : Le SLM reçoit un prompt structuré en deux parties : un **system prompt** définissant le rôle et les contraintes du modèle, et un **user prompt** fournissant dynamiquement l'index des tables, les schémas complets et la question.

System Prompt — Pandas

```
You are a Python/Pandas expert. Output ONLY code.
RULES: Data is in 'dfs' list. No file I/O.
TABLE SELECTION: Use TABLE INDEX to pick dfs[i].
Add comment: # Using dfs[i] because <reason>
OUTPUT: End with: result = {'type': ..., 'value':
...}
```

User Prompt — Pandas

```
=== TABLE INDEX ===
<dfs[0] → table1, dfs[1] → table2...>
=== FULL SCHEMAS WITH SAMPLE DATA ===
<schéma, données exemples, valeurs uniques>
=== QUESTION ===
<question en langage naturel>
=== TASK ===
1. Identify dfs[i] via index. 2. Use pd.merge() if
needed.
3. Output ONLY code. 4. End with result dict.
IMPORTANT: No file I/O (pd.read, sqlite3, open).
```

3. **Exécution et validation** : Le code Python généré est extrait du bloc ```` ```python``` ````, nettoyé puis validé syntaxiquement à l'aide de `ast.parse()`. Avant son exécution, une étape de vérification recherche l'utilisation de fonctions interdites (`open()`, `sqlite3`, `pd.read_*()`, etc.). L'exécution est ensuite réalisée dans un environnement Python contrôlé par l'application. Bien que cette approche réduise les risques liés à l'exécution de code généré automatiquement, elle ne constitue pas une sandbox de sécurité complète au sens des environnements d'exécution isolés.

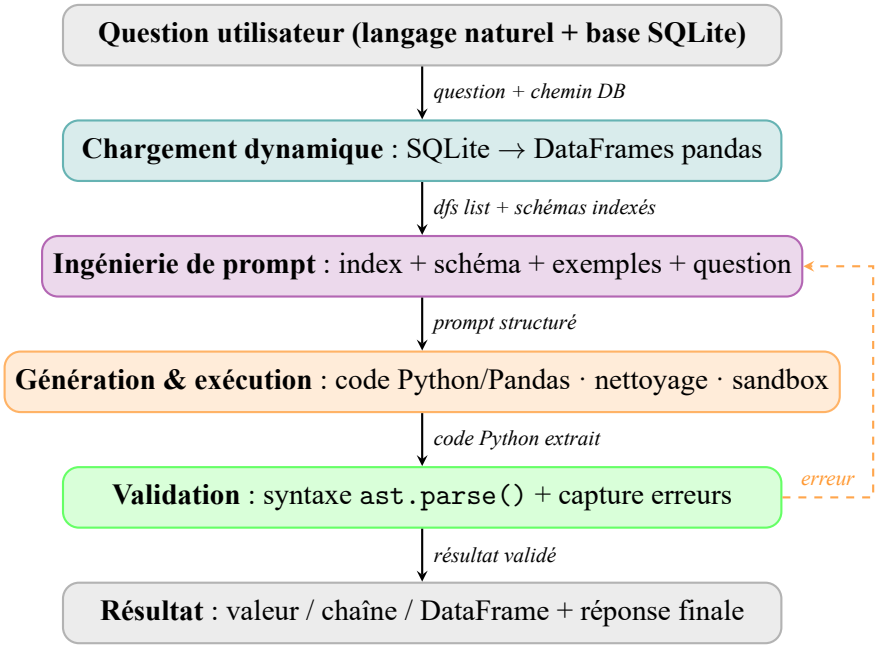


FIG. 3.3 : Pipeline pour la génération de code Python/Pandas dans ASK-DATA.

3.6.4 Architecture SLM avec génération en langue arabe

Pour l'expérience en langue arabe :

1. Les 400 questions du benchmark ont été traduites via un script Python automatisé, puis validées et raffinées par **Claude Sonnet (Anthropic, version de juin 2026)** afin de garantir une précision linguistique et technique des requêtes SQL.

2. Les mêmes schémas de bases de données ont été utilisés (en anglais, les tables et colonnes Spider étant en anglais).

3. Le prompt a été adapté pour instruire le modèle en arabe.

4. La réponse attendue est une requête SQL correcte (le SQL reste invariant, seule la compréhension de la question change).

3.6.5 Configuration des expériences comparatives

Pour garantir la comparabilité des résultats entre les architectures :

- Les mêmes 400 questions ont été utilisées pour toutes les expériences.
- Les mêmes trois modèles ont été utilisés pour toutes les architectures avancées.
- Un protocole d'évaluation identique pour toutes les architectures assure la neutralité et la fiabilité de nos mesures comparatives.
- Les paramètres de génération (température, tokens max) ont été maintenus constants.
- Toutes les expériences ont été réalisées sur la même plateforme matérielle pour chaque comparaison.

3.7 Métriques d'évaluation

3.7.1 Exact Match (EM)

L'**Exact Match** (EM) est une métrique binaire qui mesure si la requête SQL générée est *exactement identique* à la requête de référence (*gold query*). Dans notre étude, une version simplifiée de l'Exact Match est utilisée, basée sur une normalisation lexicale des requêtes SQL (mise en minuscules des mots-clés, suppression des espaces superflus).

$$\text{EM} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[\hat{q}_i = q_i^*]$$

où $\hat{q}_i$ est la requête générée pour la question i, q_i^* est la requête de référence, et N est le nombre total de questions. Cette métrique est connue pour être sévère : elle ne récompense pas des requêtes équivalentes mais syntaxiquement différentes.

3.7.2 Execution Accuracy (EX)

L'**Execution Accuracy** (EX) est une métrique plus tolérante qui mesure si l'exécution de la requête générée sur la base de données produit le *même résultat* que l'exécution de la requête de référence.

$$\text{EX} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}[f(\hat{q}_i, D_i) = f(q_i^*, D_i)]$$

où $f(q, D)$ désigne l'exécution de la requête q sur la base de données D_i. Cette métrique est plus proche de la réalité pratique : deux requêtes différentes peuvent produire des résultats

identiques. Cependant, l'Execution Accuracy peut être sensible à la taille et au contenu des bases de données testées, certaines requêtes incorrectes pouvant produire accidentellement le même résultat sur des données particulières. Elle est calculée via un script Python dédié qui exécute les requêtes sur les bases SQLite de Spider et compare les résultats.

3.7.3 Qualité linguistique des réponses

Pour évaluer la précision des requêtes SQL et la logique de raisonnement des modèles, nous avons implémenté une approche d'évaluation automatisée par **LLM-as-a-Judge**. **Claude Sonnet (Anthropic, version de juin 2026)** a été utilisé comme juge automatique avec un **prompt strict** qui inclut des règles métier et lui confère le rôle d'expert en SQL et en évaluation de modèles Text-to-SQL. Ce juge est chargé d'évaluer chaque réponse selon les critères suivants :

- **Résultat correct** (Oui/Non/Partiel) : Cette évaluation détermine si l'explication ou la requête générée correspond fidèlement à l'intention logique du SQL attendu (*ground truth*).
- **Type d'erreur** : En cas d'échec, le modèle catégorise précisément la nature de l'erreur afin de permettre une analyse diagnostique fine :
  - *Syntaxe* : Erreurs de forme (parenthèses, guillemets, virgules).
  - *Jointure* : Défaut dans la table ou la condition de liaison.
  - *Logique* : Erreurs dans les clauses *WHERE*, *GROUP BY*, *ORDER BY* ou *HAVING*.
  - *Nom de Colonne/Table* : Référence à des entités inexistantes dans le schéma.
  - *Génération* : Absence de production d'un code SQL valide.

Ce processus d'évaluation automatisé garantit une standardisation du jugement sur l'ensemble des 400 questions du benchmark, tout en assurant une traçabilité rigoureuse des points de défaillance de chaque architecture.

3.7.4 Temps d'inférence

Le **temps d'inférence** moyen par question (exprimé en millisecondes, ms) a été mesuré pour chaque modèle et chaque architecture afin d'évaluer leur efficacité opérationnelle. Pour garantir une précision optimale, chaque mesure est calculée individuellement pour chacune des 400 questions, selon deux indicateurs distincts :

- **Temps de génération SQL** : Cet indicateur mesure exclusivement la durée nécessaire pour produire la requête SQL, du lancement du prompt jusqu'à la fin de l'inférence. Il reflète la capacité intrinsèque du modèle à raisonner sur le schéma de données.

- **Temps total (Génération + Résultat)** : Ce second indicateur mesure le temps écoulé de la soumission de la question jusqu'à l'obtention finale des résultats. Il cumule le temps de génération SQL avec la latence liée à l'exécution de la requête sur la base de données SQLite et à la récupération des résultats. Pour les architectures complexes, ce temps agrège l'ensemble des étapes de traitement nécessaires à la finalisation de la réponse.
- Pour l'architecture multi-agents, nous avons calculé le temps d'inférence de l'ensemble des agents.

Cette distinction permet d'isoler la latence propre au modèle (raisonnement) de la latence système (exécution et affichage), offrant une vision granulaire de la performance des architectures, notamment pour celles intégrant des étapes de traitement supplémentaires ou une exécution sécurisée (*sandbox*).

3.7.5 Évaluation qualitative

Une évaluation qualitative manuelle a été réalisée sur un sous-ensemble de 100 exemples représentatifs (répartis équitablement selon les niveaux de complexité) pour valider les résultats de l'évaluation automatique.

3.8 Validité expérimentale

3.8.1 Reproductibilité des expériences

Pour garantir la reproductibilité :

- La température de génération a été fixée à 0 (déterministe).
- Les versions exactes de tous les modèles (fichiers GGUF) sont consignées.
- Le code source des scripts d'évaluation est structuré et documenté.
- Les résultats bruts (requêtes générées, scores) sont sauvegardés en **Excel** pour chaque expérience.

3.8.2 Menaces à la validité

Validité interne : Plusieurs facteurs peuvent influencer les performances observées indépendamment des capacités intrinsèques des modèles évalués. Tout d'abord, l'approche d'évaluation automatisée par LLM-as-Judge peut introduire un biais lié au modèle utilisé comme juge, à son paramétrage ou à son interprétation des réponses générées. Afin de limiter ce risque, un prompt strict et standardisé a été conçu pour réduire la subjectivité du jugement.

Par ailleurs, les modèles étudiés n'utilisent pas tous le même niveau de quantification GGUF (Q4_K_S, Q4_K_M, Q5_K_M, Q6_K et Q8_0). Or, la quantification influence simultanément la précision des prédictions et les performances d'inférence (temps de réponse, consommation mémoire et utilisation GPU). Une partie des écarts observés entre modèles peut ainsi être attribuable non seulement à leurs architectures ou à leurs données d'entraînement, mais également au niveau de quantification appliqué. Cette hétérogénéité constitue un facteur de confusion susceptible d'affecter la comparabilité directe des résultats.

Enfin, les modèles ont été exécutés selon différentes configurations matérielles (CPU, Full GPU ou mode hybride CPU/GPU). Les stratégies d'offloading et les limitations de la mémoire vidéo disponible (4 Go de VRAM) peuvent également influencer les temps d'inférence et, dans certains cas, les performances globales du système.

Validité externe : Les résultats obtenus sur le benchmark Spider ne peuvent pas être généralisés automatiquement à l'ensemble des bases de données réelles. Bien que Spider constitue la référence académique la plus utilisée pour l'évaluation des systèmes Text-to-SQL cross-domain, les bases de données industrielles présentent souvent des schémas plus volumineux, des données plus bruitées et des exigences métier spécifiques. De plus, l'évaluation en langue

arabe repose sur des questions traduites à partir de l'anglais. Malgré un processus de validation et de raffinement des traductions, certaines ambiguïtés linguistiques ou pertes de nuances sémantiques peuvent subsister et influencer la compréhension des questions par les modèles.

Validité de construction : L'Exact Match (EM) est une métrique particulièrement stricte qui peut sous-estimer les performances réelles des modèles, puisque deux requêtes syntaxiquement différentes peuvent être sémantiquement équivalentes. À l'inverse, l'Execution Accuracy (EX), bien que plus proche des usages pratiques, peut produire des faux positifs lorsque deux requêtes différentes génèrent accidentellement le même résultat sur certaines bases de données. Enfin, l'évaluation qualitative par LLM-as-Judge demeure une approximation du jugement humain et ne remplace pas entièrement une expertise manuelle exhaustive des requêtes SQL produites.

3.9 Conclusion

Ce chapitre a décrit l'ensemble du dispositif expérimental mis en place : deux plate-formes matérielles (CPU et GPU) exploitant LM Studio pour un fonctionnement 100% local, 400 questions du benchmark Spider couvrant tous les niveaux de complexité, dix modèles SLM aux caractéristiques variées, et un protocole d'évaluation en deux phases (sélection des meilleurs SLM puis évaluation des architectures avancées). Les métriques retenues (EM, EX, LLM-as-Judge, temps) permettent une comparaison multi-dimensionnelle des approches. Le chapitre suivant présente et analyse les résultats obtenus lors de ces expérimentations.

Chapitre 4

Résultats expérimentaux et analyse comparative

Introduction

L'avènement des modèles de langage compacts (Small Language Models – SLM) offre des perspectives inédites pour le déploiement d'assistants intelligents en environnement local, garantissant à la fois la confidentialité des données, une faible latence et une maîtrise des coûts computationnels. Dans ce paysage technologique, la tâche de génération de requêtes SQL à partir du langage naturel constitue un défi scientifique et technique majeur. Elle exige non seulement une compréhension sémantique fine de l'intention utilisateur, mais également une rigueur syntaxique et logique absolue pour interagir de manière fiable avec des bases de données relationnelles.

Cependant, l'optimisation de ces systèmes se heurte à plusieurs verrous technologiques persistants. La génération de requêtes impliquant des jointures multiples, des agrégations imbriquées ou des sous-requêtes complexes demeure une difficulté centrale pour les architectures actuelles. De plus, la littérature scientifique interroge l'efficacité réelle des lois d'échelle (*scaling laws*) dans ce domaine spécifique : l'augmentation brute du nombre de paramètres ou de la fenêtre de contexte est-elle plus déterminante qu'un fine-tuning spécialisé sur des corpus SQL ? Enfin, la robustesse de ces modèles face à des formulations ambiguës ou dans un contexte multilingue, notamment pour la langue arabe, reste un champ d'investigation critique pour les applications grand public.

Ce chapitre a pour objectif de mener une évaluation empirique exhaustive afin d'identifier les modèles et les paradigmes architecturaux les plus aptes à répondre aux exigences de performance, de robustesse et d'efficacité d'un assistant Text-to-SQL local de production (projet ASK-DATA). Pour ce faire, l'analyse s'articule autour de deux axes complémentaires et progressifs :

1. **La Phase I : Sélection des meilleurs Small Language Models** : Cette phase consiste en un benchmark comparatif de dix modèles de langage, évalués sur des plateformes CPU et GPU. L'analyse se fonde sur des métriques quantitatives rigoureuses, notamment l'Exact Match (EM), l'Execution Accuracy (EX), le taux de requêtes exécutables, la robustesse face aux hallucinations de schéma, ainsi que les temps d'inférence.

2. **La Phase II : Évaluation des architectures Text-to-SQL** : S'appuyant sur les trois modèles les plus prometteurs identifiés précédemment, cette section explore l'impact de stratégies d'amélioration avancées. Elle examine successivement l'apport du Retrieval-Augmented Generation (RAG) et l'optimisation de ses paramètres de récupération, l'efficacité de l'orchestration Multi-Agent pour la décomposition des tâches, et enfin, la capacité de généralisation des modèles à travers une analyse comparative inter-langues (corpus arabe versus corpus anglais).

3. **La Phase III : Analyse comparative globale** : Cette dernière phase synthétise l'ensemble des résultats obtenus au cours des expérimentations précédentes. Elle propose une comparaison globale des performances de tous les modèles et architectures évalués, suivie d'une analyse synthétique des temps d'inférence afin de mesurer le compromis entre efficacité computationnelle et précision. Enfin, elle examine l'impact de la complexité des requêtes sur les modèles Text-to-SQL, en distinguant les requêtes simples des requêtes complexes à jointures multiples et sous-requêtes imbriquées.

En fin, cette étude vise à démontrer que la performance optimale en Text-to-SQL local ne repose pas sur une course à la taille des modèles, mais sur une adéquation scientifique rigoureuse entre la spécialisation des données d'entraînement, la stratégie d'enrichissement contextuel et les contraintes matérielles du déploiement.

Phase I : Sélection des meilleurs Small Language Models

4.1 Résultats sur plateforme CPU

4.1.1 Analyse globale des performances

Tab. 4.1 : Performances globales des modèles sur CPU

Rang	Modèle	EM (%)	EX (%)	Exécutable (%)
1	XiYanSQL-QwenCoder-7B	**53.50**	**80.25**	**89.50**
2	SLM-SQL-1.5B	26.75	69.50	84.75
3	Granite 4.0 Tiny	28.25	65.50	82.00
4	Qwen3.5-4B	30.50	59.25	74.75
5	Qwen3.5-9B	33.00	59.00	65.75
6	Ministral-3-3B	1.50	49.50	71.50
7	SQLCoder-7B-2	0.50	46.75	69.00
8	DeepSeek-R1-7B	13.25	34.50	40.50
9	DeepSeek-R1-1.5B	8.75	25.00	33.75
10	Gemma2B Text-to-SQL	8.50	22.75	39.00

Hiérarchisation des Performances : L'examen des données révèle une stratification nette des modèles selon leurs performances. Le modèle `xiyansql-qwencoder-7b` se distingue significativement avec un score EM de 53,50 %, devançant le deuxième modèle classé (Qwen3.5-9B, 33,00 %) de plus de 20 points. Cet écart substantiel suggère une spécialisation avancée de ce modèle pour la tâche d'interprétation SQL. Les modèles granite-4.0-tiny (28,25 %) et SLM-SQL-1.5B (26,75 %) occupent respectivement les quatrième et cinquième positions en EM, démontrant une efficacité remarquable compte tenu de leur architecture légère (1,5 milliard de paramètres). Ils se classent néanmoins en deuxième et troisième positions selon la métrique EX (69,50 % et 65,50 % respectivement).

Corrélation entre Taille des Modèles et Performances : Une observation centrale de cette étude concerne la relation non linéaire entre la paramétrisation des modèles et leurs performances. Les données empiriques contredisent l'hypothèse selon laquelle une augmentation de la taille du modèle entraîne systématiquement une amélioration des résultats. En effet :

- Le modèle SLM-SQL-1.5B surpasse nettement les deux variantes DeepSeek-R1 testées

(8,75 % et 13,25 % en EM) respectivement pour les versions 1.5B et 7B, contre 26,75 % pour SLM-SQL-1.5B, malgré une taille comparable à la plus petite.

- Ces résultats suggèrent que les mécanismes de raisonnement généralistes de la famille DeepSeek-R1 ne compensent pas l'absence d'un entraînement spécifique au domaine SQL, dont les scores restent inférieurs à 15 % en EM.

Analyse de la Validité Syntaxique versus Justesse Sémantique : La comparaison différentielle entre les métriques Executable_pct et EX_pct permet d'identifier des profils comportementaux distincts parmi les modèles évalués :

- **Profil A – Génération syntaxiquement valide mais sémantiquement imprécise :** Les modèles `ministral-3-3b` et `sqlcoder-7b-2` présentent un taux d'exécutabilité élevé (71,50 % et 69,00 % respectivement) couplé à des scores EM particulièrement bas (1,50 % et 0,50 %). Cet écart de plus de 20 points entre Executable_pct et EX_pct indique une capacité à produire du code SQL syntaxiquement correct, mais dont la logique ne correspond pas à l'intention exprimée en langage naturel.
- **Profil B – Génération précise mais syntaxiquement fragile :** Le modèle Qwen3.5-9B présente un écart de 6,75 points entre Executable_pct (65,75 %) et EX_pct (59,00 %), suggérant que les requêtes valides produites sont majoritairement correctes. La limitation principale réside dans la fréquence des erreurs syntaxiques empêchant l'exécution.
- **Profil C – Équilibre entre validité et justesse :** Les modèles `xiyansql-qwencoder-7b`, `SLM-SQL-1.5B` et `granite-4.0-tiny` maintiennent des écarts modérés (entre 6 et 9 points) entre les deux métriques, témoignant d'une capacité à générer simultanément du code exécutable et sémantiquement pertinent.

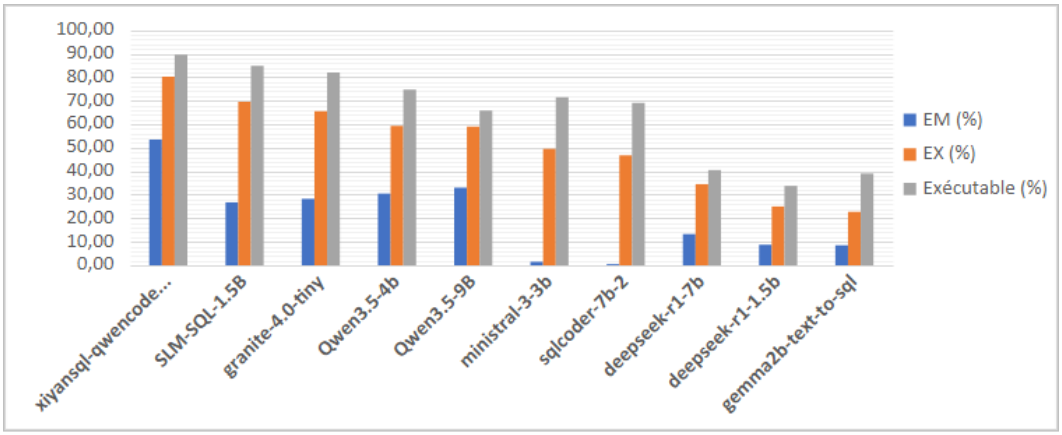


FIG. 4.1 : Les performances des 10 modèles sur CPU.

4.1.2 Analyse des performances CPU

TAB. 4.2 : Statistiques des temps de génération SQL sur CPU (en ms)

Modèle	Moyenne	Médiane	Std	Min	Max	95e percentile
Qwen3.5-4B	43 305,90	35 654,80	28 545,00	11 179,90	209 685,50	87 834,10
Qwen3.5-9B	258 682,50	224 308,60	143 667,80	75 949,30	1 599 195,60	454 197,80
SLM-SQL-1.5B	3 525,20	2 882,20	2 726,30	835,10	29 987,20	6 516,00
DeepSeek-R1-1.5B	91 052,80	61 964,40	359 713,90	25 083,50	6 678 423,40	81 624,40
DeepSeek-R1-7B	161 882,60	124 394,90	133 607,70	42 349,30	1 332 983,70	302 026,20
Gemma2B-text-to-SQL	14 796,30	8 092,50	33 580,10	3 427,90	348 380,60	31 416,70
Granite-4.0-Tiny	8 338,40	7 892,40	2 799,90	3 751,20	22 218,00	13 893,50
Ministral-3B	14 026,10	12 355,60	8 704,70	2 511,20	71 583,60	28 281,20
SQLCoder-7B-2	20 051,30	16 892,40	13 470,90	5 594,30	145 838,90	38 015,60
XiYanSQL-QwenCoder-7B	9 252,30	6 989,90	6 610,30	2 733,80	68 007,60	19 469,80

Trois groupes apparaissent :

- **Groupe 1 : Très efficaces** (SLM-SQL-1.5B – Granite 4.0 Tiny – XiYanSQL-QwenCoder-7B)
  Ils offrent un excellent compromis entre rapidité et qualité.
- **Groupe 2 : Performances intermédiaires** (Ministral – Gemma2B – SQLCoder)
  Temps raisonnables mais précision limitée.
- **Groupe 3 : Trop coûteux en CPU** (Qwen3.5-4B – Qwen3.5-9B – DeepSeek)
  Le coût d'inférence devient difficilement acceptable pour un assistant local.

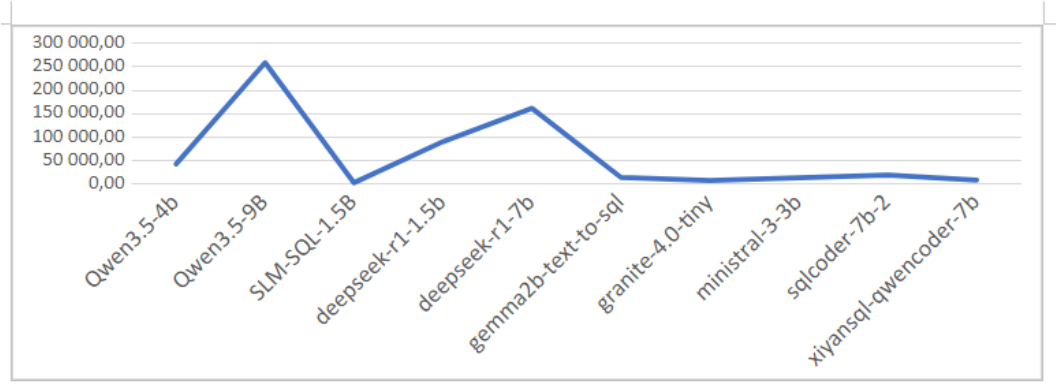


FIG. 4.2 : La moyenne des temps de formulation des réponses des 10 modèles sur CPU.

4.1.3 Analyse de la robustesse

Hallucinations de schéma : Les meilleurs modèles sont :

Tab. 4.3 : Taux d'erreurs de schéma sur CPU

Modèle	Erreur schéma (%)
Qwen3.5-9B	0.00
Qwen3.5-4B	0.75
XiYanSQL-QwenCoder-7B	3.25
SLM-SQL-1.5B	5.50
IBM Granite 4.0 Tiny	10.75

Les modèles Qwen présentent la meilleure compréhension des schémas. Cependant, XiYanSQL conserve un excellent équilibre entre robustesse et précision.

Erreurs de génération : Les modèles spécialisés SQL génèrent généralement moins de texte parasite. Particulièrement :

- XiYanSQL-QwenCoder-7B
- SLM-SQL-1.5B
- SQLCoder-7B-2
- Gemma2B Text-to-SQL Expert

Cette caractéristique est importante pour les systèmes Text-to-SQL automatisés.

4.1.4 Analyse par catégories de requêtes

- **Requêtes simples :** XiYanSQL atteint environ 85 % d'EM et plus de 93 % d'EX. Tous les modèles généralistes réussissent correctement cette catégorie.
- **Requêtes avec filtrage :** XiYanSQL obtient près de 97,5 % d'EX. La compréhension des clauses WHERE apparaît comme l'un de ses principaux points forts.
- **Requêtes d'agrégation :** Les modèles spécialisés SQL dominent : XiYanSQL-QwenCoder-7B – SLM-SQL-1.5B – IBM Granite 4.0 Tiny.
- **Requêtes ambiguës :** XiYanSQL conserve le meilleur taux de réussite (≈85,7 % EX). Cela suggère une meilleure capacité à interpréter les formulations imprécises.
- **Requêtes complexes :** Cette catégorie reste difficile pour tous les modèles. Néanmoins XiYanSQL demeure le meilleur avec environ 54 % d'EX.

4.1.5 Tendances observées

✓**Importance du fine-tuning SQL**

Les résultats montrent que les modèles spécifiquement entraînés pour la génération SQL obtiennent généralement de meilleures performances que les modèles généralistes. XiYanSQL-QwenCoder-7B et SLM-SQL-1.5B figurent parmi les modèles les plus performants de l'étude malgré une taille relativement modeste.

Cette observation confirme que la spécialisation Text-to-SQL apporte davantage de bénéfices qu'une simple augmentation du nombre de paramètres.

✓**La taille du modèle n'explique pas seule les performances**

Par exemple, XiYanSQL-QwenCoder-7B (7B paramètres) obtient de meilleurs résultats que Qwen3.5-9B (9B paramètres), malgré une taille inférieure. Les performances semblent davantage liées à la qualité du fine-tuning et aux données d'entraînement qu'au nombre de paramètres.

✓ **Impact du contexte long**

Les modèles disposant d'une fenêtre de contexte de 128K tokens ne dominent pas systématiquement les autres modèles. Les résultats suggèrent que, pour cette tâche, la spécialisation SQL et la qualité de l'entraînement jouent un rôle plus important que la longueur du contexte.

4.1.6 Recommandations pour ASK-DATA

TAB. 4.4 : Modèles recommandés – Plateforme CPU

Modèle principal recommandé	**Alternative légère**	**Alternative Edge**
XiYanSQL-QwenCoder-7B Points forts : • Meilleur EM • Meilleur EX • Meilleur taux de requêtes exécutables • Très bonne robustesse • Temps CPU raisonnable C'est le meilleur candidat pour l'approche SLM seule.	**SLM-SQL-1.5B** Points forts : • Temps d'inférence le plus faible • Très bon EX • Faible consommation mémoire • Idéal pour les machines modestes.	**IBM Granite 4.0 Tiny** Points forts : • Très bon compromis qualité/ressources • Très bonne exécutabilité • Architecture moderne.

TAB. 4.5 : Modèles à écarter – Plateforme CPU

DeepSeek-R1-Distill-Qwen-7B	**DeepSeek-R1-Distill-Qwen-1.5B**	**Gemma2B Text-to-SQL Expert**	**SQLCoder-7B-2**
Très lent, précision insuffisante.	Faible qualité SQL.	Hallucinations importantes. EX faible.	Résultats inférieurs aux attentes.

Conclusion

L'analyse complète des **10 modèles** montre que **XiYanSQL-QwenCoder-7B** est le meilleur modèle pour un assistant local Text-to-SQL. Il domine simultanément les métriques de précision, d'exécution et de robustesse tout en conservant un coût CPU raisonnable. Les résultats démontrent également que la **spécialisation Text-to-SQL est plus déterminante que la taille du modèle ou la longueur du contexte**, ce qui constitue une conclusion scientifique importante pour la suite du projet ASK-DATA.

4.2 Résultats sur plateforme GPU

4.2.1 Analyse académique des performances globales

Résultats globaux :

TAB. 4.6 : Performances globales des 10 modèles sur GPU

Rang	Modèle	EM (%)	EX (%)	SQL exécutable (%)
1	XiYanSQL-QwenCoder-7B	**52.75**	**79.75**	**89.00**
2	Qwen3.5-9B	32.75	55.00	59.50
3	Granite 4.0 Tiny	28.25	65.75	81.50
4	SLM-SQL-1.5B	26.25	69.00	85.25
5	Qwen3.5-4B	17.25	25.75	23.50
6	DeepSeek-R1-7B	11.25	33.00	40.00
7	Gemma2B Text-to-SQL	9.00	23.75	39.25
8	DeepSeek-R1-1.5B	8.00	24.00	30.50
9	Ministral 3B	1.50	50.25	72.00
10	SQLCoder-7B-2	0.50	45.50	68.25

- **XiYanSQL-QwenCoder-7B obtient les meilleures performances sur l'ensemble des métriques.**
  - +20 points d'EM par rapport au deuxième.
  - Taux d'exécution SQL proche de 90 %.
- **SLM-SQL-1.5B réalise une performance remarquable compte tenu de sa taille.**
  - Seulement 1,5B paramètres.
  - EX = 69 %.
  - Exécutabilité = 85,25 %.
- **Granite 4.0 Tiny constitue un résultat notable.**
  - Modèle Edge.
  - EM supérieur à plusieurs modèles 7B.
  - Excellent taux d'exécution.
- **SQLCoder-7B-2 présente un comportement atypique.**
  - Très faible EM.
  - EX relativement élevé.
  - Le modèle génère souvent des requêtes exécutables mais non identiques à la référence.

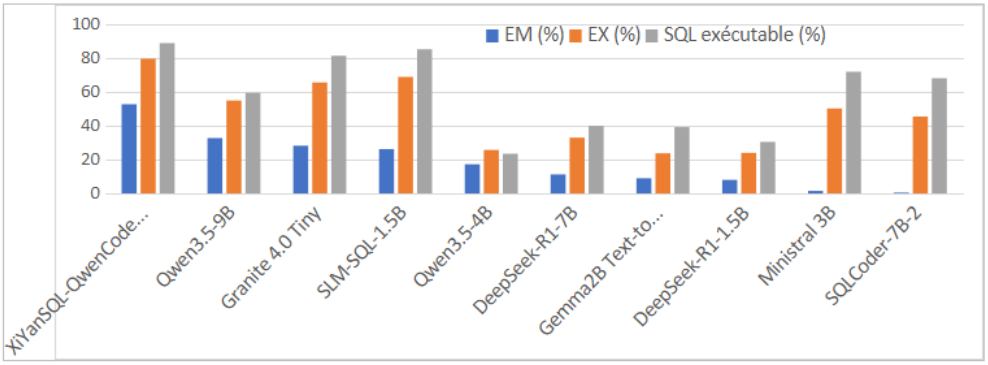


FIG. 4.3 : Les performances des 10 modèles sur GPU.

Analyse des temps de réponse :

TAB. 4.7 : Temps de réponse des modèles sur GPU

Modèle	Temps moyen (ms)	P95 (ms)
SLM-SQL-1.5B	**1 076**	**2 086**
Gemma2B Text-to-SQL	1 515	2 956
Ministral 3B	1 776	4 401
XiYanSQL-QwenCoder-7B	3 916	7 839
Granite 4.0 Tiny	9 287	16 574
SQLCoder-7B-2	9 366	19 414
DeepSeek-R1-1.5B	34 242	55 213
Qwen3.5-4B	45 109	67 890
DeepSeek-R1-7B	71 603	141 669
Qwen3.5-9B	163 233	287 794

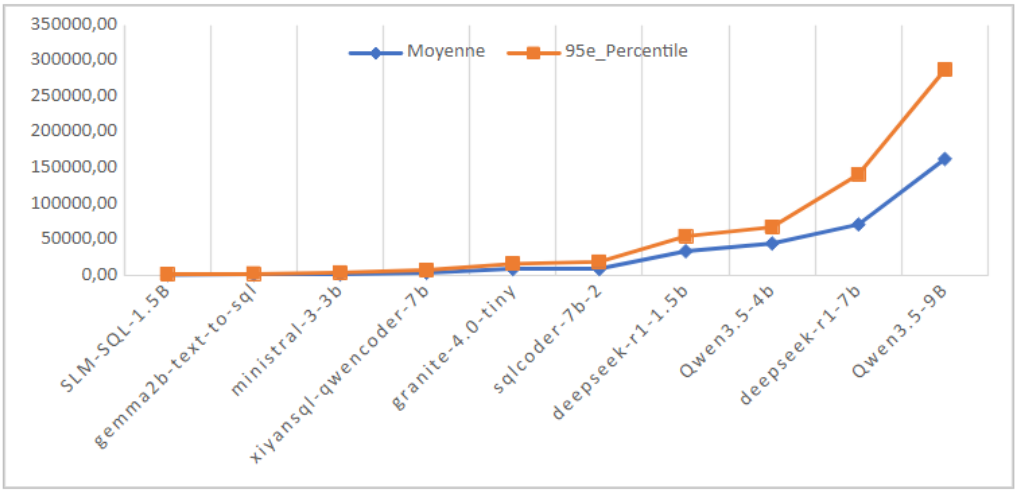


FIG. 4.4 : Les mesures de temps des 10 modèles sur GPU.

Les modèles de raisonnement généraliste (Qwen3.5 et DeepSeek-R1) sont extrêmement coûteux en temps de génération. Pour un assistant local interactif : un temps < 2 s est excellent, 2–5 s est acceptable, 10 s est pénalisant, 30 s est problématique. Seuls SLM-SQL-1.5B, Gemma2B, Ministral et XiYanSQL offrent une expérience réellement fluide.

Analyse selon la complexité :

TAB. 4.8 : Performances selon la complexité des requêtes sur GPU

Requêtes simples (EX)		**Requêtes moyennes (EX)**	
Modèle	%	Modèle	%
XiYanSQL	87.9	XiYanSQL	52.4
SLM-SQL	78.3	Granite	39.0
Granite	73.6	SLM-SQL	36.6
		Ministral	34.1
Requêtes complexes : EM=0%, EX=0%			

La complexité SQL reste le principal verrou scientifique. Les modèles rencontrent encore de grandes difficultés avec : sous-requêtes imbriquées, agrégations multiples, jointures complexes, clauses HAVING avancées.

Analyse des jointures :

Lorsque le nombre de JOIN augmente :

- baisse systématique de l'EM.
- baisse plus lente de l'EX.

Les modèles les plus robustes sont :

- XiYanSQL-QwenCoder-7B
- SLM-SQL-1.5B
- Granite 4.0 Tiny

Les modèles généralistes perdent rapidement en précision lorsque plusieurs tables interviennent.

4.2.2 Analyse comparative orientée Text-to-SQL

Les modèles SQL spécialisés sont-ils supérieurs ?

Globalement : **oui**. Le *fine-tuning* SQL apporte un avantage net et décisif sur les capacités de raisonnement des modèles généralistes.

Tab. 4.9 : Comparaison modèles spécialisés vs généralistes (EX %)

Modèles SQL spécialisés		Modèles généralistes	
Modèle	**EX (%)**	**Modèle**	**EX (%)**
XiYanSQL	79.75	Qwen3.5-9B	55.00
SLM-SQL	69.00	DeepSeek-R1-7B	33.00
SQLCoder	45.50	Qwen3.5-4B	25.75
Gemma2B SQL	23.75		

Le contexte long aide-t-il ?

Les résultats suggèrent : **Non, pas de façon significative.** Le gain provient davantage du fine-tuning SQL, de la qualité des données d'entraînement, de la spécialisation.

Tab. 4.10 : Contexte long vs EX

Modèle	Contexte	EX (%)
XiYanSQL	64K	79.75
SLM-SQL	32K	69.00
Qwen3.5-9B	128K	55.00

Robustesse face aux hallucinations :

Le taux de SQL exécutable constitue l'indicateur le plus direct de la robustesse d'un modèle face aux hallucinations. Un SQL non exécutable traduit en effet une erreur générée par le modèle : référence à une colonne ou une table inexistante, violation de syntaxe, ou incohérence avec le schéma fourni. Plus ce taux est élevé, moins le modèle hallucine.

Tab. 4.11 : Taux de SQL exécutable – Robustesse GPU

Modèle	Exécutable (%)
XiYanSQL	89.00
SLM-SQL	85.25
Granite	81.50
Ministral	72.00
SQLCoder	68.25

4.2.3 Forces et faiblesses par famille

TAB. 4.12 : Forces et faiblesses des familles de modèles

Famille	Forces	Faiblesses
Modèles SQL spécialisés	Meilleure précision, moins d'hallucinations, gestion des jointures.	Moins polyvalents, moins robustes hors domaine SQL.
Modèles généralistes	Bon raisonnement, compréhension langage naturel.	Temps de réponse élevés, erreurs SQL fréquentes.
Modèles Edge	Très faible coût matériel, excellente vitesse.	Précision limitée sur requêtes complexes.

4.2.4 Recommandations pour le développement d'ASK-DATA

TAB. 4.13 : Recommandations pour le développement d'ASK-DATA

Recommandation n°1 XiYanSQL-QwenCoder-7B	**Recommandation n°2 SLM-SQL-1.5B**	**Recommandation n°3 Granite 4.0 Tiny**
Pourquoi ? - EM = 52.75% - EX = 79.75% - SQL exécutable = 89% - Temps moy. ≈ 3.9 s *Meilleur compromis précision /robustesse.* **Cas d'usage :** Version principale de l'assistant ASK-DATA.	**Pourquoi ?** - EX = 69% - SQL exécutable = 85.25% - Temps moy. ≈ 1.1 s - Taille ≈ 1.9 GB *Excellent pour :* portables, mini-PC, assistants embarqués.	**Pourquoi ?** - EX = 65.75% - SQL exécutable = 81.50% - ≈ 1.4B paramètres. *Très bon compromis Edge.*

TAB. 4.14 : Modèles à écarter – GPU

Modèle	Justification
SQLCoder-7B-2	EM faible (0.5%), performances insuffisantes.
Gemma2B Text-to-SQL	Performances insuffisantes.
DeepSeek-R1-1.5B	Faible précision.
DeepSeek-R1-7B	Trop lent pour le gain obtenu.
Qwen3.5-4B	Précision modeste.
Ministral 3B	EX correct mais EM très faible.

Conclusion

Cette campagne GPU montre clairement que **la spécialisation SQL est plus déterminante que la taille du modèle ou la longueur du contexte**. Le modèle **XiYanSQL-QwenCoder-7B** s'impose comme la référence de l'étude, tandis que **SLM-SQL-1.5B** apparaît comme le meilleur compromis pour un déploiement local léger. Enfin, **Granite 4.0 Tiny** démontre que les modèles Edge modernes peuvent rivaliser avec certains modèles généralistes beaucoup plus volumineux.

Phase II : Évaluation des architectures Text-to-SQL

Introduction

La première partie de cette étude, intitulée **« Sélection des meilleurs Small Language Models »**, a permis d'évaluer les performances de dix modèles de langage compacts (SLM) dans une tâche de génération de requêtes SQL à partir de questions formulées en langage naturel. Cette phase comparative a porté sur plusieurs critères d'évaluation, notamment le taux d'Exact Match, la validité syntaxique des requêtes SQL, les performances selon les catégories de questions ainsi que les temps d'exécution sur une architecture CPU et une architecture GPU.

À l'issue de cette première campagne expérimentale, trois modèles se sont distingués par leurs performances globales et leur adéquation avec les contraintes d'un assistant Text-to-SQL local : **SLM-SQL-1.5B**, **IBM Granite 4.0 Tiny** et **XiYanSQL-QwenCoder-7B**. Ces modèles ont ainsi été retenus pour les expérimentations approfondies présentées dans cette seconde partie.

La Phase II s'appuie directement sur les résultats obtenus lors de la phase de sélection des SLM. Son objectif est d'étudier différentes stratégies d'amélioration des performances des assistants Text-to-SQL locaux en exploitant les modèles les plus prometteurs identifiés précédemment. Cette partie est organisée autour de trois axes complémentaires.

Le premier axe concerne l'évaluation de l'approche **SLM + RAG (Retrieval-Augmented Generation)**. Cette étude vise à mesurer l'apport d'un mécanisme de récupération documentaire sur la qualité des requêtes SQL générées et à déterminer les paramètres de récupération les plus adaptés aux architectures retenues.

Le deuxième axe porte sur l'analyse académique des résultats de l'architecture **Multi-Agent**. L'objectif est d'évaluer l'intérêt d'une décomposition de la tâche Text-to-SQL entre plusieurs agents spécialisés afin d'améliorer le raisonnement, la compréhension des requêtes et la génération SQL finale.

le troisième axe propose une analyse comparative des SLM pour un assistant Text-to-SQL local appliqué à un corpus **arabe**. Cette expérimentation permet d'étudier la capacité des modèles sélectionnés à traiter des requêtes rédigées en arabe et d'évaluer leur robustesse dans un contexte multilingue, particulièrement important pour les applications locales destinées aux utilisateurs arabophones.

Enfin, Le quatrième axe s'intéresse à l'évaluation des approches de **génération et d'exécution de code Python** basées sur des Small Language Models. Deux stratégies sont examinées : d'une part, le couplage **SLM + Pandas**, où le modèle génère directement du code Python de manipulation de données via la bibliothèque Pandas, et d'autre part, l'approche **PandasAI**, qui abstrait cette génération au travers d'une interface conversationnelle. L'ob-

jectif est de mesurer la viabilité de ces alternatives à la génération SQL classique pour l'interrogation de données tabulaires en langage naturel.

Ainsi, cette seconde partie ne se limite pas à comparer des modèles de langage, mais vise à analyser l'impact de différentes architectures d'assistance – RAG, Multi-Agent et traitement multilingue – sur les performances globales d'un système Text-to-SQL local.

4.3 Résultats de l'approche SLM + RAG

4.3.1 Analyse de l'influence des paramètres de récupération sur les performances du système RAG

Objectif de l'expérimentation : Cette expérimentation a pour objectif d'identifier la configuration optimale des paramètres de récupération d'un système RAG (Retrieval-Augmented Generation) destiné à la génération automatique de requêtes SQL. L'évaluation a été réalisée sur un corpus de 400 questions couvrant différents niveaux de complexité et plusieurs catégories de requêtes.

Trois modèles de langage représentatifs ont été retenus pour cette étude :

- XiYanSQL-QwenCoder-7B ;
- SLM-SQL-1.5B ;
- IBM Granite 4.0 Tiny.

Les expérimentations portent principalement sur deux paramètres clés du module de recherche vectorielle :

- La **métrique de similarité** utilisée pour la récupération documentaire : Cosine Similarity, Inner Product (IP) et Distance Euclidienne (L2).
- Le **seuil de similarité** appliqué lors de la sélection des documents pertinents : 0.1, 0.2, 0.3 et 0.4 (pour Cosine et IP) ; 0.5, 0.8, 1.0 et 1.2 (pour L2).

L'évaluation repose sur plusieurs indicateurs quantitatifs, notamment le taux d'Execution Accuracy (EX), le pourcentage de requêtes SQL valides, les temps de traitement (récupération et génération), ainsi que les performances obtenues selon les différentes catégories de questions (voir le tableau 4.15).

TAB. 4.15 : Influence des paramètres de récupération RAG sur l'Execution Accuracy (%)

Métrique	Seuil	Granite 4.0 Tiny	SLM-SQL-1.5B	XiYanSQL-QwenCoder-7B
Cosine	0,1	53,75	46,25	72,75
Cosine	0,2	54,75	46,25	72,50
Cosine	0,3	54,25	44,75	69,50
Cosine	0,4	46,50	35,75	54,25
IP	**0,1**	53,75	**46,50**	**73,00**
IP	**0,2**	**55,00**	46,25	72,50
IP	0,3	54,25	44,75	69,50
IP	0,4	46,50	35,75	54,25
L2	0,5	0	0	0
L2	0,8	0	0	0
L2	1,0	0	0	0
L2	1,2	0	0	0

Influence de la métrique de recherche : Les résultats obtenus mettent en évidence une quasi-équivalence entre les métriques **Cosine Similarity** et **Inner Product** pour l'ensemble des modèles évalués. Les écarts observés restent très faibles et ne dépassent pas 0,25 point d'Execution Accuracy. Cette proximité des performances suggère que les vecteurs d'embedding utilisés sont déjà normalisés, conduisant ainsi les deux métriques à sélectionner des ensembles de documents très similaires.

À l'inverse, l'utilisation de **la distance euclidienne (L2)** conduit à un échec systématique du processus de génération. Aucun des modèles testés ne parvient à produire des requêtes SQL correctes dans cette configuration. En conséquence, la métrique L2 apparaît inadaptée au système RAG étudié et ne devrait pas être retenue dans la configuration finale.

Influence du seuil de similarité : Les meilleures performances sont obtenues avec des seuils de similarité faibles, compris entre 0.1 et 0.2. Lorsque le seuil augmente, une diminution progressive des performances est observée : **XiYanSQL-QwenCoder-7B** enregistre une baisse d'environ 18,75 points de pourcentage, tandis que les pertes atteignent 10,75 points pour **SLM-SQL-1.5B** et 8,50 points pour **IBM Granite 4.0 Tiny**.
Cette dégradation s'explique par la réduction du nombre de documents récupérés. Un seuil plus élevé limite la quantité d'informations transmises au modèle de génération, ce qui peut conduire à un contexte incomplet et à l'absence de certaines informations nécessaires à la construction correcte des requêtes SQL. Ces observations confirment qu'un seuil trop restrictif pénalise fortement l'efficacité globale du système RAG.

Comparaison des modèles : En considérant la meilleure configuration de récupération pour chaque modèle, le classement global obtenu est présenté dans le Tableau 4.16

TAB. 4.16 : Meilleure Execution Accuracy obtenue par modèle en configuration RAG optimale

Modèle	Execution Accuracy (%)
XiYanSQL-QwenCoder-7B	73.00
IBM Granite 4.0 Tiny	55.00
SLM-SQL-1.5B	46.50

Le modèle spécialisé **XiYanSQL-QwenCoder-7B** se distingue nettement des autres approches en obtenant le meilleur taux d'Execution Accuracy. Il dépasse **IBM Granite 4.0 Tiny** de 18 points et **SLM-SQL-1.5B** de 26,5 points.

Cette supériorité confirme l'intérêt des modèles spécifiquement entraînés pour les tâches Text-to-SQL. Le fine-tuning spécialisé permet une meilleure compréhension du schéma de données, des relations entre tables et des constructions SQL complexes, même lorsque le système repose sur un mécanisme RAG.

Conclusion : Les résultats obtenus montrent que les performances d'un système RAG dépendent fortement du paramétrage du module de récupération documentaire. Les métriques **Cosine Similarity** et **Inner Product**, associées à des seuils faibles compris entre 0,1 et 0,2, constituent la configuration la plus efficace pour l'ensemble des modèles évalués. À l'inverse, la distance **L2** s'avère inadaptée et conduit à un échec complet du processus de génération.

Par ailleurs, le modèle **XiYanSQL-QwenCoder-7B** apparaît comme la solution la plus performante avec un taux d'Execution Accuracy de **73 %** et un taux de **100 %** de requêtes SQL valides, démontrant l'efficacité des modèles spécialisés SQL dans un contexte RAG. Ces résultats soulignent également que l'optimisation de la phase de récupération documentaire est un facteur aussi important que le choix du modèle de génération pour la conception d'un assistant Text-to-SQL performant.

4.3.2 Analyse des résultats de l'approche SLM + RAG

L'ensemble des tests a été réalisé sur une plateforme d'exécution basée exclusivement sur le processeur (CPU), en utilisant la métrique **Inner Product (IP)** avec un **seuil de similarité fixé à 0,1**, cette combinaison ayant démontré les meilleures performances lors des phases préliminaires d'évaluation.

Analyse académique des résultat :

Performances globales :

Tab. 4.17 : Performances globales – SLM + RAG

Modèle	EM (%)	EX (%)	Requêtes exécutables (%)
SLM-SQL-1.5B	20,50	71,50	82,75
Granite 4.0 Tiny	31,25	68,25	87,25
XiYanSQL-QwenCoder-7B	**56,75**	**83,25**	**93,25**

Le modèle **XiYanSQL** domine très nettement les deux autres :

- $+25{,}5$ points d'EM par rapport à **Granite**
- $+36{,}25$ points d'EM par rapport à **SLM-SQL**

L'écart est particulièrement important sur la métrique **Exact Match (EM)**, qui mesure la capacité à produire exactement la requête SQL attendue. Pour un assistant Text-to-SQL réel, cette métrique est essentielle car elle reflète la compréhension correcte du schéma et de l'intention utilisateur.

Distribution des erreurss :

Tab. 4.18 : Distribution des erreurs – SLM + RAG

Modèle	Sans erreur (%)	Erreur jointure (%)	Erreur logique (%)
SLM-SQL-1.5B	74.75	4.75	20.50
Granite 4.0 Tiny	74.75	1.50	23.75
XiYanSQL-QwenCoder-7B	**88.00**	**1.75**	**10.25**

Les erreurs logiques constituent le principal facteur de dégradation (mauvaise colonne sélectionnée, GROUP BY inutile, agrégation incorrecte, mauvaise interprétation de la question). XiYanSQL réduit pratiquement de moitié ces erreurs. Cela indique un apprentissage plus robuste des structures SQL.

Tab. 4.19 : Temps de réponse – SLM + RAG (CPU)

Modèle	Moyenne (ms)	Médiane (ms)	P95 (ms)
SLM-SQL-1.5B	**16 168**	**5 392**	66 222
Granite 4.0 Tiny	23 612	19 758	**41 959**
XiYanSQL-QwenCoder-7B	35 767	10 765	157 706

Temps de réponse :

Tab. 4.20 : Interprétation des temps de réponse RAG

SLM-SQL	Granite	XiYanSQL
Le plus rapide. Idéal pour matériel modeste, exécution CPU-only, faible consommation mémoire.	**Stabilité.** Meilleur P95, comportement prévisible.	**Qualité.** Plus lent, mais produit les requêtes les plus précises. Pour un assistant local professionnel, quelques secondes supplémentaires sont souvent acceptables si le SQL généré est correct.

Analyse par catégorie de questions :

Tab. 4.21 : Performances par catégorie de questions – SLM + RAG

	Exact Match (%)				Execution Accuracy (%)		
Catégorie	**SLM-SQL**	**Granite**	**XiYanSQL**	**Catégorie**	**SLM-SQL**	**Granite**	**XiYanSQL**
Simple	46.25	71.25	**87.50**	Simple	83.75	90.00	**93.75**
Filtrage	20.00	26.25	**77.50**	Filtrage	93.75	85.00	**96.25**
Agrégation	11.25	27.50	**40.00**	Agrégation	65.00	53.75	**76.25**
Ambiguë	27.14	35.71	**68.57**	Ambiguë	68.57	77.14	**82.86**
Complexe	1.11	0.00	**16.67**	Complexe	48.89	40.00	**68.89**

Impact de la complexité SQL :

Tab. 4.22 : Impact de la complexité SQL – SLM + RAG

	Exact Match (%)				Execution Accuracy (%)		
Complexité	**SLM-SQL**	**Granite**	**XiYanSQL**	**Complexité**	**SLM-SQL**	**Granite**	**XiYanSQL**
Simple	26.11	39.81	**68.79**	Simple	78.03	76.75	**87.58**
Moyen (1 JOIN)	0	0	**13.41**	Moyen	48.78	39.02	**65.85**
Complexe	0	0	0	Complexe	25.00	0	**100**

Analyse comparative orientée Text-to-SQL :

Classement global :

Quel modèle offre le meilleur compromis ?

TAB. 4.23 : Classement global des modèles

Rang	Modèle	Commentaire
1	**XiYanSQL-QwenCoder-7B**	Meilleure précision SQL
2	**Granite 4.0 Tiny**	Bon compromis taille/robustesse
3	**SLM-SQL-1.5B**	Très léger mais limité

Forces et faiblesses des modèles :

- **XiYanSQL-QwenCoder-7B**
  - *Forces :* Meilleur EM (56,75 %) et EX (83,25 %), réduction des erreurs logiques, performance optimale sur les filtres complexes et les jointures.
  - *Faiblesses :* Temps d'inférence plus élevé et empreinte mémoire plus importante.
- **IBM Granite 4.0 Tiny**
  - *Forces :* Format très compact, stabilité temporelle, bonne gestion générale des requêtes et peu d'erreurs de jointure.
  - *Faiblesses :* Difficultés marquées sur le SQL complexe et les opérations d'agrégation.
- **SLM-SQL-1.5B**
  - *Forces :* Extrêmement léger, grande rapidité d'exécution, idéal pour un déploiement CPU.
  - *Faiblesses :* EM faible, erreurs logiques fréquentes et gestion lacunaire des jointures complexes.

Robustesse face aux schémas complexes : L'analyse qualitative des questions difficiles montre que :

TAB. 4.24 : Robustesse qualitative face aux schémas complexes – RAG

XiYanSQL	Granite	SLM-SQL
Produit généralement : • les bonnes tables • les bonnes jointures • les bonnes colonnes Les requêtes générées sont souvent identiques à la référence.	Tendance observée : • hallucination de tables intermédiaires • jointures supplémentaires non nécessaires Exemple : Ajout d'une table `model_list` inexistante.	Tendance observée : • ajout de GROUP BY inutile • réécriture excessive • erreurs de logique métier

Tendances scientifiques observées :

✓ **La spécialisation SQL est déterminante :** Le modèle spécialisé SQL (XiYanSQL) surpasse largement Granite (Edge) et SLM-SQL (compact), malgré une taille seulement modérément supérieure.
Conclusion : Le fine-tuning SQL apporte davantage que l'augmentation brute de paramètres.

✓ **La taille seule ne suffit pas :** Granite possède un contexte 128K et une architecture moderne. Pourtant : EM = 31,25 % contre 56,75 % pour XiYanSQL.
Le contexte long n'améliore donc pas automatiquement le Text-to-SQL.

✓ **Les JOIN restent le principal défi :** Tous les modèles chutent lorsque plusieurs tables interviennent et que les relations deviennent implicites. Même XiYanSQL montre une baisse significative.

Recommandations pour le développement d'ASK-DATA :

TAB. 4.25 : Recommandations pour le développement d'ASK-DATA – RAG

Modèle recommandé n°1 XiYanSQL-QwenCoder-7B	**Modèle recommandé n°2 Granite 4.0 Tiny**	**Modèles à écarter SLM-SQL-1.5B**
Justification : EM = 56.75% ; EX = 83.25% ; Exécutable = 93.25% ; Erreurs logiques 10.25%. Il apparaît comme le candidat le plus pertinent de l'étude.	Alternative intéressante lorsque : • RAM limitée • déploiement edge • CPU uniquement	À éviter comme moteur principal. Motifs : • EM très faible (20.5%) • nombreuses erreurs logiques • incapacité à gérer les jointures Peut rester utile comme baseline académique ou prototype léger.

Les résultats montrent clairement que **XiYanSQL-QwenCoder-7B est le candidat le plus adapté au développement d'un assistant Text-to-SQL local de type ASK-DATA**. Sa spécialisation SQL lui permet d'obtenir un avantage majeur sur toutes les métriques importantes (EM, EX, robustesse et gestion des jointures). Les modèles compacts tels que Granite 4.0 Tiny restent intéressants pour les environnements très contraints, mais ne peuvent actuellement rivaliser avec un modèle spécifiquement entraîné pour la génération SQL.

4.4 Analyse académique des résultats Multi-Agent

4.4.1 Analyse des performances globales

Tab. 4.26 : Performances globales – Architecture Multi-Agent

Modèle	EM (%)	EX (%)	Requêtes exécutables (%)
SLM-SQL-1.5B	23.50	66.00	85.25
granite-4.0-tiny	24.75	60.50	82.50
xiyansql-qwencoder-7b	**51.75**	**73.75**	**86.75**

Le modèle **XiYanSQL-QwenCoder-7B** domine clairement l'évaluation : +28,25 points d'EM par rapport à **SLM-SQL-1.5B**, +27,00 points d'EM par rapport à **Granite 4.0 Tiny**, +7,75 points d'EX face à **SLM-SQL-1.5B**, et le meilleur taux d'exécutabilité avec **86,75 %**. L'écart est particulièrement important sur l'Exact Match, ce qui indique une meilleure compréhension du schéma relationnel, des jointures, des agrégations et des conditions complexes.

4.4.2 Analyse des erreurs

Tab. 4.27 : Distribution des erreurs – Architecture Multi-Agent

Modèle	Aucune err. (%)	Génér. (%)	Joint. (%)	Logiq. (%)
SLM-SQL-1.5B	66.25	6.00	4.50	23.25
granite-4.0-tiny	61.25	6.00	2.75	30.00
xiyansql-qwencoder-7b	**76.75**	6.00	**1.75**	**15.50**

Granite Tiny : Les erreurs logiques atteignent **30 %**, soit le pire score. Le principal problème est la mauvaise sélection des colonnes, les erreurs d'interprétation de la question et le choix incorrect de tables.

SLM-SQL-1.5B : Comportement plus stable mais encore limité : bonnes requêtes simples, difficultés sur les raisonnements multi-tables.

XiYanSQL : Réduit fortement les erreurs de jointure et les erreurs logiques. C'est généralement l'indicateur le plus important pour un assistant Text-to-SQL réel.

4.4.3 Analyse des temps de réponse

Tab. 4.28 : Temps de réponse – Architecture Multi-Agent

Modèle	Temps moyen (ms)	Médiane (ms)	95^{e} percentile (ms)
SLM-SQL-1.5B	**8 822**	**5 280**	40 808
granite-4.0-tiny	27 762	21 983	97 493
xiyansql-qwencoder-7b	14 154	8 320	**39 187**

Surprise importante : Malgré ses 7B paramètres, **XiYanSQL est presque deux fois plus rapide que Granite Tiny.** Cela suggère une meilleure optimisation, une architecture plus efficace et une spécialisation SQL réduisant le besoin de raisonnement inutile.

4.4.4 Analyse par catégorie de questions

Tab. 4.29 : Performances par catégorie – Multi-Agent

	Exact Match (%)				Execution Accuracy (%)		
Catégorie	**SLM-SQL**	**Granite**	**XiYanSQL**	**Catégorie**	**SLM-SQL**	**Granite**	**XiYanSQL**
Agrégation	13.75	17.50	**41.25**	Agrégation	63.75	45.00	**70.00**
Ambiguë	25.71	24.29	**58.57**	Ambiguë	58.57	61.43	**65.71**
Complexe	1.11	0.00	**15.56**	Complexe	56.67	46.67	**72.22**
Filtrage	27.50	28.75	**72.50**	Filtrage	72.50	73.75	**78.75**
Simple	52.50	56.25	**76.25**	Simple	78.75	77.50	**81.25**

- **Questions simples :** Les trois modèles restent compétitifs.
- **Questions complexes :** La supériorité du modèle se révèle ici de manière flagrante. Même si la performance absolue (15,56 %) traduit les limites actuelles des systèmes face à une haute complexité, XiYanSQL demeure la seule approche présentant une véritable capacité de généralisation et de résolution pour ce type de requêtes.

4.4.5 Analyse par complexité SQL

Tab. 4.30 : Impact de la complexité SQL – Multi-Agent

	Exact Match (%)				Execution Accuracy (%)		
Complexité	**SLM-SQL**	**Granite**	**XiYanSQL**	**Complexité**	**SLM-SQL**	**Granite**	**XiYanSQL**
Simple	29.94	31.53	**62.74**	Simple	69.11	64.97	**74.52**
Moyen	0	0	**10.98**	Moyen	54.88	45.12	**70.73**
Complexe	0	0	**25.00**	Complexe	50.00	25.00	**75.00**

Observation majeure : Les deux petits modèles échouent totalement en Exact Match dès qu'une jointure est présente. XiYanSQL conserve 11 % d'EM sur les requêtes moyennes et 25 % sur les requêtes complexes, ce qui constitue une différence qualitative et non seulement quantitative.

4.4.6 Analyse des jointures

TAB. 4.31 : Impact du nombre de jointures – Multi-Agent

Exact Match (%)				Execution Accuracy (%)			
Nombre de JOIN	**SLM-SQL**	**Granite**	**XiYanSQL**	**Nombre de JOIN**	**SLM-SQL**	**Granite**	**XiYanSQL**
0	29.94	31.53	**62.74**	0	69.11	64.97	**74.52**
1	0	0	**14.29**	1	66.07	48.21	**78.57**
2	0	0	**9.09**	2	45.45	50.00	**77.27**

Les jointures constituent le principal facteur discriminant. Pour un assistant Text-to-SQL réel : les requêtes mono-table sont rares ; les requêtes métier comportent souvent plusieurs jointures. XiYanSQL est nettement supérieur sur ce point.

4.4.7 Analyse qualitative des questions difficiles

TAB. 4.32 : Analyse qualitative – Multi-Agent

SLM-SQL-1.5B	**Granite Tiny**	**XiYanSQL**
Tendance à : inventer des tables ; modifier la structure SQL attendue ; sur-complexifier certaines requêtes simples.	Tendance à : halluciner des schémas inexistants ; remplacer les tables du schéma par des tables imaginaires. Exemple observé : utilisation de `players`, `rankings`, `model_list`, etc. alors que ces tables n'existent pas dans le schéma cible.	Produit généralement : les bonnes tables ; les bonnes colonnes ; les bonnes jointures. Le comportement est beaucoup plus robuste face aux schémas.

4.4.8 Tendances observées

✓Les modèles spécialisés SQL restent largement supérieurs : Cette étude confirme ce qui est observé dans Spider et BIRD. Le fine-tuning SQL apporte davantage de bénéfices qu'une augmentation brute du contexte.

✓**Le contexte 128K n'apporte pas d'avantage visible :** Granite dispose de 128K tokens. Pourtant : EM inférieur ; EX inférieur ; plus d'erreurs logiques. Le facteur dominant est donc la spécialisation SQL et non la fenêtre de contexte.

✓**La taille seule ne suffit pas :** SLM-SQL et Granite ont des tailles proches. Pourtant SLM-SQL obtient un meilleur EX. L'entraînement spécialisé SQL reste le facteur déterminant.

4.4.9 Recommandations pour un assistant Text-to-SQL local

Tab. 4.33 : Recommandations – Architecture Multi-Agent

Recommandation n°1 : XiYanSQL-QwenCoder-7B	**Recommandation n°2 : SLM-SQL-1.5B**	**Modèle à écarter : granite-4.0-tiny**
Justification : C'est le meilleur compromis : précision ; robustesse ; vitesse acceptable ; gestion des jointures. **Usage recommandé :** Assistant Text-to-SQL principal de ASK-DATA.	**Cas d'utilisation :** PC modestes ; CPU uniquement ; prototypes embarqués. **Avantages :** très faible coût ; meilleure latence ; EX correct (66 %).	**Motifs :** EM faible ; EX le plus faible ; erreurs logiques les plus nombreuses ; latence la plus élevée. Le modèle n'apporte aucun avantage concret sur cette tâche.

Conclusion

Les résultats montrent que **la spécialisation SQL demeure le facteur dominant**. Parmi les trois modèles évalués, **XiYanSQL-QwenCoder-7B** s'impose très clairement comme le candidat le plus adapté au développement d'un assistant Text-to-SQL local, grâce à sa combinaison de précision élevée, robustesse face aux jointures et faible taux d'erreurs logiques. **SLM-SQL-1.5B** constitue une alternative intéressante pour les environnements fortement contraints en ressources, tandis que **Granite 4.0 Tiny** ne présente pas d'avantage suffisant pour justifier son intégration dans un système Text-to-SQL de production.

4.5 Analyse comparative des SLM pour un assistant Text-to-SQL local (Corpus Arabe)

4.5.1 Analyse académique des résultats

Performances globales :

Tab. 4.34 : Performances globales – Corpus Arabe

Modèle	EM (%)	EX (%)	Exécutable (%)
SLM-SQL-1.5B	18.50	49.50	83.50
Granite 4.0 Tiny	21.25	51.00	76.50
XiYanSQL-QwenCoder-7B	**38.00**	**67.00**	**88.50**

Deux observations importantes apparaissent : premièrement XiYanSQL **obtient les meilleures performances sur tous les indicateurs** : +16 points EM par rapport à Granite, +19 points EX, et meilleur taux d'exécution. Deuxièmement, l'écart entre EM et EX est très élevé pour tous les modèles, signifiant que de nombreuses requêtes produites ne sont pas identiques à la référence mais donnent néanmoins le bon résultat. Pour un assistant Text-to-SQL réel, **EX est souvent plus représentatif que EM**, car l'utilisateur cherche avant tout une réponse correcte.

Analyse des erreurs :

Tab. 4.35 : Distribution des erreurs – Corpus Arabe

Modèle	Aucune erreur (%)	Logique (%)	Jointure (%)	Syntaxe (%)
SLM-SQL	49.75	31.75	13.00	5.50
Granite	50.75	25.50	20.50	3.25
XiYanSQL	**66.75**	**17.50**	14.50	**1.25**

Les erreurs de syntaxe sont globalement faibles. Le principal problème est l'**erreur logique** (mauvais filtre, mauvaise colonne, mauvaise agrégation, mauvaise condition). XiYanSQL réduit presque de moitié ces erreurs.

Analyse par catégorie de questions :

Tab. 4.36 : Performances par catégorie – Corpus Arabe

Exact Match (%)				Execution Accuracy (%)			
Catégorie	**SLM-SQL**	**Granite**	**XiYanSQL**	**Catégorie**	**SLM-SQL**	**Granite**	**XiYanSQL**
Simple	43.75	61.25	**66.25**	Simple	60.0	82.5	**87.5**
Filtrage	16.25	10.00	**41.25**	Filtrage	67.5	61.25	**77.5**
Agrégation	10.00	11.25	**23.75**	Agrégation	51.25	35.0	**55.0**
Ambiguë	25.71	27.14	**54.29**	Ambiguë	48.57	55.71	**74.29**
Complexe	0.00	0.00	**10.00**	Complexe	23.33	24.44	**44.44**

XiYanSQL obtient **74,29 %** d'EX sur les questions ambiguës, contre 55,71 % pour **Granite** et 48,57 % pour **SLM-SQL**. Cela suggère une meilleure capacité de raisonnement sémantique face à des formulations imprécises ou ambiguës.

Impact de la complexité SQL :

Tab. 4.37 : Impact de la complexité SQL – Corpus Arabe

Exact Match (%)				Execution Accuracy (%)			
Complexité	**SLM-SQL**	**Granite**	**XiYanSQL**	**Complexité**	**SLM-SQL**	**Granite**	**XiYanSQL**
Simple	23.57	27.07	**46.82**	Simple	57.01	59.24	**73.89**
Moyen	0	0	**6.10**	Moyen	21.95	21.95	**43.90**
Complexe	0	0	0	Complexe	25	0	0

Observation majeure : Les trois modèles s'effondrent lorsque la complexité SQL augmente.

Tab. 4.38 : Résumé global – Corpus Arabe

Complexité	EM moyen (%)	EX moyen (%)
Simple	32.48	63.38
Moyen	2.03	29.27
Complexe	0.00	8.33

On observe un phénomène classique du Text-to-SQL : La génération reste correcte pour les requêtes simples mais devient très difficile dès l'apparition de plusieurs jointures et agrégations.

Analyse des jointures :

TAB. 4.39 : Impact des jointures – Corpus Arabe

	Exact Match (%)		Execution Acc. (%)			
Modèle	**Join=0**	**Join=1**	**Modèle**	**Join=0**	**Join=1**	**Join=2**
SLM-SQL	23.57	0	SLM-SQL	57.01	25.00	18.18
Granite	27.07	0	Granite	59.24	26.79	4.55
XiYanSQL	**46.82**	**8.93**	XiYanSQL	**73.89**	**42.86**	**45.45**

La gestion des jointures est probablement le principal facteur discriminant. XiYanSQL est le seul modèle montrant une capacité significative à gérer plusieurs tables.

Analyse des temps de réponse :

TAB. 4.40 : Temps de réponse – Corpus Arabe

Modèle	**Moyenne (ms)**	**Médiane (ms)**	**P95 (ms)**
SLM-SQL	**6 852**	**5 309**	16 109
Granite	9 025	8 652	**14 666**
XiYanSQL	9 436	7 834	19 398

SLM-SQL est le plus rapide. XiYanSQL est environ 38% plus lent que SLM-SQL mais produit beaucoup plus de requêtes correctes. Le gain de précision compense largement la légère augmentation de latence dans un assistant local.

4.5.2 Analyse comparative orientée Text-to-SQL

TAB. 4.41 : Meilleur compromis – Corpus Arabe

Critère	**Gagnant**
Meilleure précision	XiYanSQL
Meilleure robustesse	XiYanSQL
Meilleure gestion des jointures	XiYanSQL
Plus faible latence	SLM-SQL
Plus faible taille mémoire	Granite / SLM-SQL
Meilleur compromis global	XiYanSQL

Pour la robustesse Text-to-SQL, plusieurs indicateurs convergent plus faible taux d'erreurs logiques, meilleur EX, meilleure gestion des questions ambiguës, meilleure gestion des jointures. Le modèle le plus robuste est clairement : **XiYanSQL-QwenCoder-7B**

4.5.3 Tendances observées

✓Le fine-tuning SQL est déterminant : Les deux modèles spécialisés SQL surpassent le modèle généraliste. Même un modèle de seulement 1,5B paramètres peut rivaliser avec Granite. Cela confirme qu'en Text-to-SQL, la spécialisation est souvent plus importante que la taille brute.

✓La taille seule ne suffit pas : Granite possède 128K contexte et une architecture moderne, mais reste derrière XiYanSQL. Le contexte long n'apporte pas d'avantage visible sur ce benchmark.

✓Les requêtes complexes restent un défi : Tous les modèles montrent EM ≈ 0 % et EX très faible sur les requêtes complexes. C'est probablement la principale limite actuelle de l'assistant.

4.5.4 Recommandations pour le développement de l'assistant local

TAB. 4.42 : Recommandations finales – Corpus Arabe

Recommandation n°1 : XiYanSQL-QwenCoder-7B	**Recommandation n°2 : SLM-SQL-1.5B**	**Modèle à écarter : Granite 4.0 Tiny**
Justification : Il domine pratiquement toutes les métriques. **Cas d'usage :** Assistant Text-to-SQL principal ; déploiement local sur machine disposant de 8 à 12 Go RAM minimum ; architecture ASK-DATA en production.	**Cas d'usage :** Edge computing ; Raspberry Pi ; PC très modestes ; prototype embarqué. **Avantages :** seulement 1,9 Go ; temps de réponse le plus faible.	**Motifs :** moins précis que XiYanSQL ; plus lent que SLM-SQL ; plus faible taux d'exécution ; faibles performances sur les jointures. Il n'est jamais premier sur un critère majeur.

Conclusion

Pour un assistant Text-to-SQL local en arabe, **XiYanSQL-QwenCoder-7B apparaît nettement comme le meilleur candidat**, offrant le meilleur équilibre entre précision, robustesse et capacité à traiter des requêtes ambiguës ou multi-tables. **SLM-SQL-1.5B** constitue une excellente alternative pour les environnements très contraints en ressources, tandis que **Granite 4.0 Tiny** ne présente pas d'avantage compétitif significatif dans cette étude.

4.6 Analyse complémentaire : comparaison avec les résultats du corpus anglais

Cette analyse se concentre uniquement sur les trois modèles retenus pour la suite de l'étude. L'objectif est de comparer leur comportement sur le **corpus anglais** (Spider-like) avec les résultats précédemment observés sur le **corpus arabe**.

Comparaison globale Arabe vs Anglais :

TAB. 4.43 : Comparaison des performances Arabe vs Anglais

Modèle	EM Arabe	EM Anglais	Gain	EX Arabe	EX Anglais	Gain
SLM-SQL-1.5B	18.50	26.75	+8.25	49.50	69.50	+20.00
Granite 4.0 Tiny	21.25	28.25	+7.00	51.00	65.50	+14.50
XiYanSQL-QwenCoder-7B	38.00	**53.50**	**+15.50**	67.00	**80.25**	**+13.25**

Tous les modèles obtiennent de meilleurs résultats en anglais, cohérent avec les jeux de données d'entraînement (Spider, BIRD, WikiSQL, SQLCreateContext) qui sont majoritairement en anglais. L'écart est particulièrement marqué pour **XiYanSQL**, dont l'EM passe de 38 % à 53,5 %.

Classement général : Même sur le corpus anglais, le classement reste exactement le même que dans l'étude arabe : XiYanSQL > SLM-SQL > Granite. Cela renforce la robustesse des conclusions obtenues.

Analyse des erreurs – Corpus Anglais :

TAB. 4.44 : Répartition des erreurs – Corpus Anglais (%)

Type d'erreur	SLM-SQL	Granite	XiYanSQL
Aucune erreur	70.50	67.00	**81.50**
Jointure	9.75	5.25	5.75
Logique	13.75	17.00	**9.50**
Colonnes/Tables	5.50	10.75	**3.25**
Syntaxe	0.50	0.00	0.00

XiYanSQL Présente le moins d'erreurs logiques ; le moins d'hallucinations de colonnes ; le plus fort taux de requêtes correctes. Pour un assistant Text-to-SQL, ces deux critères sont particulièrement importants. Une erreur logique est souvent plus dangereuse qu'une erreur de syntaxe :

`SELECT SUM(price)` au lieu de : `SELECT AVG(price)`

La requête s'exécute mais fournit un résultat faux. XiYanSQL réduit fortement ce risque.

Analyse par catégorie de questions – Corpus Anglais :

TAB. 4.45 : Performances par catégorie – Corpus Anglais

	Exact Match (%)				Execution Accuracy (%)		
Catégorie	**SLM-SQL**	**Granite**	**XiYanSQL**	**Catégorie**	**SLM-SQL**	**Granite**	**XiYanSQL**
Simple	67.5	75.0	**85.0**	Simple	80.0	87.5	**93.75**
Filtrage	28.75	25.0	**71.25**	Filtrage	87.5	90.0	**97.5**
Agrégation	10.0	16.25	**37.5**	Agrégation	71.25	45.0	**73.75**
Ambiguë	31.43	27.14	**68.57**	Ambiguë	70.0	67.14	**85.71**
Complexe	0	1.11	**12.22**	Complexe	42.22	41.11	**54.44**

Analyse qualitative : Robustesse et défis techniques :

- **Robustesse face aux questions ambiguës :** Les questions ambiguës sont les plus proches d'un usage réel en langage naturel.

  "Quel est le meilleur joueur ?"
  "Quelle est la voiture la plus performante ?"

  L'écart de performance constaté est considérable. Cela démontre que **XiYanSQL** possède une meilleure capacité à interpréter les superlatifs, les critères implicites ainsi que les raisonnements complexes de classement.

- **Analyse des requêtes complexes :** Malgré ses performances, le modèle demeure perfectible sur les requêtes à haute complexité structurelle. Les requêtes impliquant des *JOIN* multiples, des clauses *GROUP BY*, *HAVING* ou des sous-requêtes constituent le défi majeur du système.

  Cette observation justifie pleinement l'exploration de pistes d'amélioration telles que :

  – L'optimisation des approches **RAG SQL** ;
  – L'implémentation de techniques de **Chain-of-Thought** ;
  – La décomposition systématique des requêtes en sous-tâches logiques.

Résultat scientifique majeur de la comparaison Arabe / Anglais :

- Les modèles spécialisés SQL restent fortement influencés par leur corpus d'entraînement anglophone.
- Le passage à l'arabe induit une perte significative de compréhension sémantique.
- Malgré cela, **XiYanSQL conserve son leadership dans les deux langues**, ce qui indique une meilleure capacité de généralisation.

4.7 Évaluation des approches de génération et d'exécution de code Python basées sur des Small Language Models

4.7.1 Analyse des résultats Pandas + SLM

Remarques :

1. **Dans cette expérimentation, seule la métrique EX (Execution Accuracy) a été retenue pour l'évaluation des performances.** En effet, l'approche étudiée repose sur la génération et l'exécution de code Python utilisant la bibliothèque Pandas, et non sur la production directe de requêtes SQL. Par conséquent, une comparaison syntaxique ou sémantique avec les requêtes SQL de référence (Gold Queries) n'est pas applicable, ce qui rend les métriques classiques d'évaluation Text-to-SQL, telles que l'Exact Match (EM), inadaptées à ce contexte.

2. **L'ensemble des expérimentations a été réalisé dans un environnement d'exécution accéléré par GPU,** afin de réduire les temps d'inférence et de garantir des conditions de calcul homogènes pour tous les modèles évalués.

Analyse globale des performances

TAB. 4.46 : Évaluation de l'exécution correcte du code Pandas généré.

Modèle	EX Correct	EX (%)	Temps moyen (ms)	Médiane (ms)	95^{e} percentile (ms)
XiYanSQL-QwenCoder-7B	**199/400**	**49,75**	31 184	21 865	77 418
Granite 4.0 Tiny	141/400	35,25	45 703	28 585	113 653
SLM-SQL-1.5B	83/400	20,75	**1 839**	**1 215**	**5 637**

Le modèle **XiYanSQL** obtient près de **50 %** d'exécution correcte, soit $+14{,}5$ points par rapport à **Granite** et $+29$ points par rapport à **SLM-SQL**. Cela montre que les compétences acquises lors du fine-tuning SQL restent largement transférables vers la génération de code Pandas.

- **SLM-SQL-1.5B** : extrêmement rapide (temps moyen de **1 839 ms**), il constitue un excellent candidat pour le prototypage rapide ou les environnements à ressources limitées.
- **XiYanSQL-QwenCoder-7B** : temps de réponse plus important (**31 184 ms** en moyenne), mais acceptable pour un assistant local fonctionnant sur GPU.
- **Granite 4.0 Tiny** : résultat surprenant – modèle beaucoup plus petit, mais plus lent que XiYanSQL (**45 703 ms**). Cela suggère que l'architecture hybride Mamba/Transformer n'apporte pas d'avantage sur cette tâche de génération Pandas.

Analyse des erreurs

Tab. 4.47 : Répartition des types d'erreurs par modèle (en %).

Type d'erreur	XiYanSQL	Granite	SLM-SQL
Aucune erreur	**49,5**	35,75	25,75
Erreur Python	17,75	**23,5**	9,75
Filtrage incorrect	0	0	**7**
Mauvaise colonne	1,25	1	1,5
Réponse vide	1,75	3	1
Résultat incomplet	4,75	3,5	**20,75**
Valeurs incorrectes	24,25	32,75	**34,25**

Tab. 4.48 : Tendances observées par modèle.

Modèle	Tendances observées
XiYanSQL-QwenCoder-7B	Principalement des erreurs de calcul ou de logique métier ; peu d'erreurs syntaxiques Python, ce qui témoigne d'une bonne maîtrise de Pandas.
Granite 4.0 Tiny	Le taux élevé d'erreurs Python (23,5 %) révèle une génération de code moins stable et des difficultés à produire des scripts exécutables.
SLM-SQL-1.5B	Nombreux résultats incomplets et erreurs de raisonnement. Le modèle semble souvent comprendre la question mais ne produire qu'une solution partielle.

Analyse par catégorie de requêtes

Tab. 4.49 : Exact Execution (%) par catégorie de requêtes.

Catégorie	XiYanSQL	Granite	SLM-SQL
Agrégation	**47,5**	25,0	2,5
Ambiguë	**41,43**	32,86	17,14
Complexe	**12,22**	4,44	10,0
Filtrage	**80,0**	61,25	22,5
Simple	**71,25**	56,25	52,5

✓ **Requêtes simples :** Les trois modèles restent relativement compétitifs.

- ✓ **Filtrage :** XiYanSQL domine très largement. Cela est cohérent avec son entraînement spécialisé en SQL.
- ✓ **Agrégations :** Différence spectaculaire ; XiYanSQL : 47.5%, Granite : 25% et SLM-SQL : 2.5%.
- ✓ **Les opérations de type** *GROUP BY*, agrégation, *COUNT*, *MEAN* et *SUM* restent difficiles pour les petits modèles.
- ✓ **Requêtes complexes :** Les performances chutent fortement pour tous les modèles. Même XiYanSQL ne dépasse que légèrement 12%.

Analyse par niveau de complexité

TAB. 4.50 : Exact Execution (%) par niveau de complexité.

Complexité	XiYanSQL	Granite	SLM-SQL
Simple	**65,22**	50,87	31,30
Moyen	**47,50**	25,00	2,50
Complexe	**12,22**	4,44	10,00

Conclusion : La difficulté de la requête reste le facteur dominant. Lorsque des opérations complexes apparaissent (jointures, regroupements, sous-requêtes, calculs multi-tables), les performances s'effondrent pour tous les modèles.

Robustesse des modèles

TAB. 4.51 : Classement et analyse de la robustesse des modèles.

Classement	Points forts	Faiblesses
1. XiYanSQL-QwenCoder-7B	Meilleur taux EX ; meilleur comportement sur les requêtes difficiles ; moins d'erreurs de génération.	Aucune faiblesse majeure signalée.
2. Granite 4.0 Tiny	Robuste sur les requêtes simples ; bonne généralisation.	Nombreuses erreurs Python.
3. SLM-SQL-1.5B	Rapidité exceptionnelle.	Faible capacité de raisonnement ; nombreux résultats incomplets.

L'expérience met clairement en évidence que :

- La spécialisation SQL est bénéfique mais insuffisante à elle seule.

- La taille du modèle reste un facteur déterminant : XiYanSQL (7B) domine, tandis que SLM-SQL (1.5B) reste limité malgré son entraînement spécialisé.

Impact de la fenêtre de contexte :

Les résultats ne montrent **aucun avantage évident** du contexte long. Pour cette tâche, les requêtes sont courtes et les schémas sont limités ; la fenêtre de contexte n'apparaît donc pas comme un facteur critique.

Conclusion :

L'étude montre que **XiYanSQL-QwenCoder-7B** est clairement le meilleur modèle pour un assistant local basé sur Pandas, avec **49,75 %** d'exécutions correctes. **Granite 4.0 Tiny** constitue le **meilleur compromis ressources/performance**, tandis que **SLM-SQL-1.5B** se distingue par sa **vitesse exceptionnelle** mais reste insuffisant pour des tâches analytiques complexes. Le principal enseignement est que, même dans un environnement Pandas, la **spécialisation SQL reste un avantage majeur**, alors que l'augmentation de la fenêtre de contexte n'apporte pas de gain significatif.

4.7.2 Analyse des résultats : Pandas-AI + SLM

Dans le cadre de cette étude, une seconde approche a été évaluée en remplaçant l'utilisation directe de la bibliothèque **Pandas** par la bibliothèque **Pandas-AI**. L'hypothèse initiale était que l'intégration de Pandas-AI pourrait simplifier le développement d'un assistant Text-to-SQL local en déléguant une partie du raisonnement à la bibliothèque.

Cependant, les expérimentations réalisées avec les différents SLM exécutés localement ont montré des résultats particulièrement limités. Aucun des modèles évalués n'a été capable de produire une réponse exploitable. Dans la totalité des tests, l'exécution s'est terminée par un dépassement du temps de réponse (**timeout**), y compris lorsque l'accélération matérielle par **GPU** était activée.

Plusieurs facteurs peuvent expliquer ces performances insuffisantes :

- **La capacité limitée de raisonnement des SLM locaux**, notamment pour les tâches nécessitant une planification multi-étape.
- **Le coût computationnel induit par les appels itératifs** et les mécanismes de réflexion interne de Pandas-AI.
- **Les contraintes de mémoire et de puissance de calcul** des environnements d'exécution locaux.
- **L'absence d'optimisation de Pandas-AI** pour des modèles quantifiés de petite taille exécutés hors des infrastructures de calcul à grande échelle.

En conséquence, pour la conception d'un assistant Text-to-SQL local destiné à des environnements à ressources contraintes, l'approche basée sur la génération directe de code Pandas par des SLM spécialisés apparaît actuellement comme une solution plus robuste, plus prévisible et mieux adaptée aux contraintes de déploiement local.

Phase III : Analyse comparative globale

Les résultats obtenus permettent d'évaluer les performances de trois SLM dans six scénarios d'utilisation différents : exécution locale sur CPU, accélération GPU, intégration d'un système RAG, architecture Multi-Agent, corpus arabe et génération de code avec Pandas/GPU. Les modèles étudiés sont : **SLM-SQL-1.5B**, **granite-4.0-tiny** et **xiyansql-qwencoder-7b**.

4.8 Comparaison globale des performances

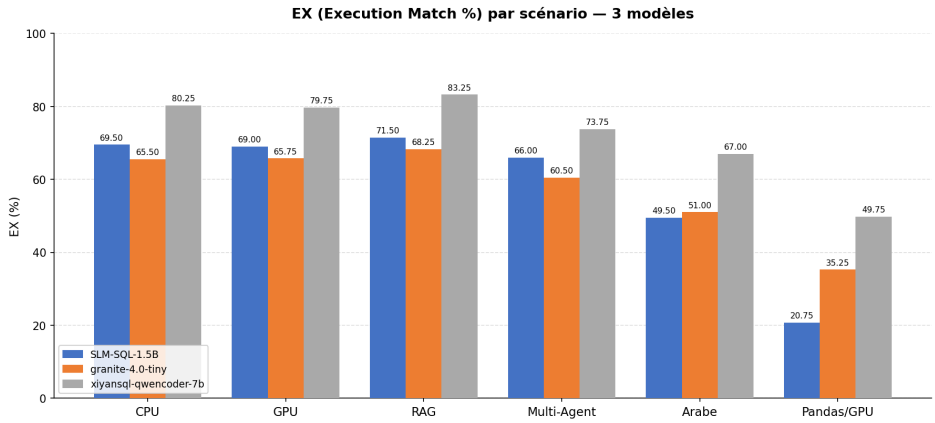


FIG. 4.5 : Évaluation de l'Execution Match (EX %) dans six scénarios.

1. **Suprématie de la spécialisation SQL et inertie matérielle**

   XiYanSQL-QwenCoder-7B domine largement les autres modèles grâce à son entraînement spécialisé, atteignant un EX moyen de **72,29 %** contre **57,71 %** pour SLM-SQL-1.5B et Granite-4.0-Tiny. Sur le plan de l'Exact Match, l'écart est encore plus marqué : **42,12 %** pour XiYanSQL contre **19,25 %** et **22,29 %** pour les deux autres modèles. L'accélération GPU n'apporte aucun gain significatif en précision (EM, EX) : les scores restent quasi-identiques entre CPU et GPU pour tous les modèles. Elle n'améliore que le temps d'inférence.

2. **Le duel architectural : RAG vs Multi-Agent**

   RAG : C'est l'amélioration la plus efficace. Elle booste les performances de Granite et XiYanSQL. La combinaison XiYanSQL + RAG atteint le meilleur score EX de l'étude avec **83,25 %**, contre **71,50 %** pour SLM-SQL-1.5B et **68,25 %** pour Granite dans le même scénario.

   Multi-Agent : Cette architecture dégrade les performances de tous les modèles par rapport au RAG. XiYanSQL perd **9,50 points** d'EX (73,75 % vs 83,25 %), SLM-SQL-1.5B perd **5,50 points** (66,00 % vs 71,50 %), et Granite perd **7,75 points** (60,50 % vs 68,25 %). Elle introduit des erreurs de propagation et des pertes d'information, s'avérant contre-productive dans ce contexte.

3. **Limites et défis spécifiques (Arabe & Pandas)**

   Corpus arabe : C'est le scénario le plus difficile. On observe une chute significative de l'EX pour tous les modèles : XiYanSQL descend à **67,00 %**, SLM-SQL-1.5B à **49,50 %** et Granite à **51,00 %**, traduisant l'impact des ambiguïtés linguistiques et du manque de données d'entraînement arabes. XiYanSQL conserve néanmoins la première place.

   Pandas + GPU : La spécialisation SQL de XiYanSQL ne se transfère pas complètement au raisonnement procédural (Python/DataFrames). Si XiYanSQL reste le meilleur avec **49,75 %** d'EX, l'écart avec Granite (**35,25 %**) se réduit, et SLM-SQL-1.5B n'atteint que **20,75 %**. L'EM est nul pour tous les modèles dans ce scénario, confirmant que le paradigme de génération de code Pandas est fondamentalement différent de la génération SQL.

4. **Conclusion scientifique et recommandation**

   La spécialisation du modèle est le facteur clé de succès. Pour développer un assistant Text-to-SQL local optimal, l'étude recommande sans équivoque l'architecture **XiYanSQL-QwenCoder-7B + RAG**, offrant le meilleur compromis entre précision sémantique (EX : 83,25 %) et robustesse d'exécution. Le Multi-Agent et le support multilingue (arabe) restent des défis ouverts nécessitant des recherches approfondies.

4.9 Analyse synthétique des temps d'inférence

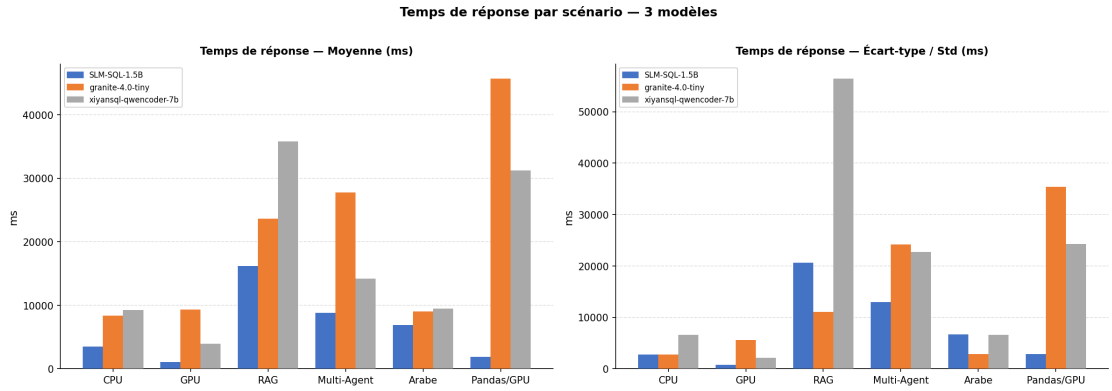


FIG. 4.6 : Temps de réponse moyen et écart-type (ms) par scénario pour les trois modèles.

L'analyse des temps d'exécution met en évidence des comportements distincts entre les trois modèles selon les architectures évaluées.

En mode **CPU**, **SLM-SQL-1.5B** présente le temps moyen le plus faible (**3 525 ms**) et la meilleure stabilité (σ = 2 726 ms), ce qui en fait le modèle le plus adapté aux environnements à ressources limitées. **Granite-4.0-Tiny** affiche un temps intermédiaire (8 338 ms, σ = 2 800 ms). À l'inverse, **XiYanSQL-QwenCoder-7B** est le plus coûteux en calcul (9 252 ms) et présente la plus forte variabilité temporelle (σ = 6 610 ms).

L'accélération **GPU** améliore significativement les performances de **SLM-SQL-1.5B** (de 3 525 ms à 1 076 ms, gain d'environ **3,3**×) et de **XiYanSQL-QwenCoder-7B** (de 9 252 ms à 3 916 ms, gain d'environ **2,4**×), tout en réduisant leur variabilité. En revanche, **Granite-4.0-Tiny** ne bénéficie pas de l'utilisation du GPU : son temps d'inférence passe de 8 338 ms à 9 287 ms, soit une légère dégradation (**gain négatif de 0,9**×).

L'intégration du **RAG** entraîne une augmentation importante de la latence pour tous les modèles. **SLM-SQL-1.5B** passe à 16 168 ms (σ = 20 630 ms), **Granite-4.0-Tiny** à 23 612 ms (σ = 11 025 ms), et **XiYanSQL-QwenCoder-7B** à 35 767 ms avec une très forte dispersion (σ = 56 469 ms), indiquant une sensibilité élevée à la taille et à la complexité du contexte récupéré. Cette surcharge s'explique par les étapes supplémentaires de récupération d'information et d'augmentation du contexte.

L'architecture **Multi-Agent** constitue une approche particulièrement coûteuse pour **Granite-4.0-Tiny** (27 762 ms, σ = 24 171 ms). **XiYanSQL-QwenCoder-7B** s'en sort mieux avec 14 154 ms (σ = 22 740 ms), et **SLM-SQL-1.5B** reste le plus rapide à 8 823 ms (σ = 12 954 ms). Les écarts-types élevés observés pour tous les modèles suggèrent une forte variabilité du nombre d'interactions entre agents et une faible prédictibilité des temps de réponse.

Les tests sur le **corpus arabe** produisent des temps intermédiaires et relativement stables :

SLM-SQL-1.5B (6 852 ms, σ = 6 661 ms), Granite-4.0-Tiny (9 025 ms, σ = 2 845 ms) et XiYanSQL-QwenCoder-7B (9 436 ms, σ = 6 634 ms), traduisant un surcoût lié aux opérations de désambiguïsation linguistique et au raisonnement multilingue.

Enfin, le scénario **Pandas + GPU** révèle un comportement particulier : **SLM-SQL-1.5B** est de loin le plus rapide (1 839 ms, σ = 2 879 ms), tandis que **Granite-4.0-Tiny** atteint son pic de latence (45 703 ms, σ = 35 428 ms). **XiYanSQL-QwenCoder-7B** se situe à 31 184 ms (σ = 24 306 ms).

En résumé, SLM-SQL-1.5B offre le meilleur compromis entre rapidité et stabilité sur CPU et Pandas/GPU, alors que XiYanSQL-QwenCoder-7B fournit les meilleures performances en précision Text-to-SQL au prix d'un coût computationnel et d'une variabilité temporelle plus élevés, particulièrement visible dans les architectures RAG et Multi-Agent.

4.10 Impact de la complexité sur les modèles Text-to-SQL

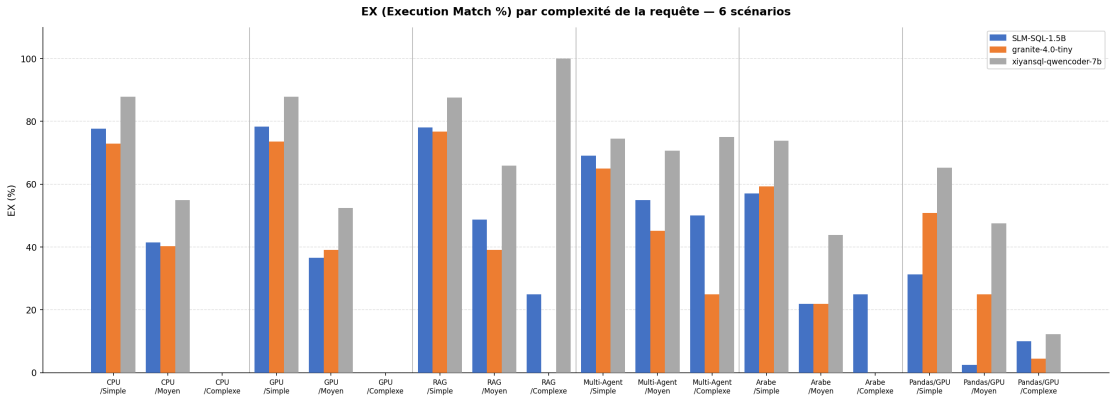


FIG. 4.7 : Évaluation EX par complexité de la requête (Simple / Moyen / Complexe) dans six scénarios.

- **Constat général**

  Les performances des modèles chutent drastiquement avec la complexité des requêtes dans tous les scénarios. En mode CPU et GPU (sans aide externe), XiYanSQL-QwenCoder-7B atteint **87,90 %** d'EX sur les requêtes simples, contre **77,71 %** pour SLM-SQL-1.5B et **72,93 %** pour Granite. Dès que la complexité augmente (requêtes moyennes avec JOIN), les scores chutent : **54,88 %** pour XiYanSQL, **41,46 %** pour SLM-SQL-1.5B et **40,24 %** pour Granite en mode CPU. Sur les requêtes complexes (multi-JOIN), le taux d'EX tombe à **0 %** pour tous les modèles en CPU et GPU, confirmant l'absence de capacité de raisonnement complexe sans architecture d'aide.

- **Apport des architectures avancées**

RAG : Le RAG améliore significativement les requêtes de complexité moyenne. XiYanSQL atteint **65,85 %** sur le niveau Moyen (contre 54,88 % en CPU) et surtout **100 %** sur les requêtes complexes, un score remarquable. SLM-SQL-1.5B progresse à **48,78 %** sur le Moyen et **25 %** sur le Complexe. Granite stagne à **39,02 %** sur le Moyen et **0 %** sur le Complexe, montrant ses limites structurelles même avec enrichissement contextuel.

Multi-Agent : C'est la solution la plus équilibrée pour les requêtes complexes : XiYanSQL atteint **75 %** d'EX sur le Complexe, SLM-SQL-1.5B **50 %** et Granite **25 %**. Sur les requêtes Moyennes, XiYanSQL monte à **70,73 %**. Cependant, cette architecture dégrade les scores sur les requêtes simples : XiYanSQL perd 13 points (74,52 % vs 87,90 %), illustrant l'effet de sur-ingénierie de la décomposition multi-agent.

- **Limites et cas particuliers**

  Corpus arabe : La chute de performance est systématique sur toutes les complexités. Sur les requêtes simples, XiYanSQL tombe à **73,89 %** (contre 87,90 % en CPU), SLM-SQL-1.5B à **57,01 %** et Granite à **59,24 %**. Sur les requêtes moyennes, les scores s'effondrent à **43,90 %**, **21,95 %** et **21,95 %** respectivement. Les requêtes complexes restent inaccessibles (**0 %**) pour Granite et XiYanSQL, et seulement **25 %** pour SLM-SQL-1.5B.

  Pandas + GPU : Le comportement est radicalement différent des scénarios SQL classiques. Sur les requêtes simples, XiYanSQL-QwenCoder-7B obtient **65,22 %** d'EX, Granite **50,87 %** et SLM-SQL-1.5B **31,30 %**. Les requêtes moyennes (agrégations) voient une chute importante : **47,50 %**, **25,00 %** et **2,50 %**. Sur les requêtes complexes (JOIN/sous-requête), les trois modèles peinent : XiYanSQL à **12,22 %**, SLM-SQL-1.5B à **10,00 %** et Granite à **4,44 %**, confirmant que la génération de code Pandas complexe reste un défi majeur même pour des modèles spécialisés SQL.

Discussion générale

La présente étude a permis d'évaluer de manière exhaustive les performances de divers Small Language Models (SLM) et architectures d'inférence dans le cadre de la génération de requêtes SQL à partir du langage naturel (Text-to-SQL). L'analyse approfondie des résultats expérimentaux, menée sur des plateformes CPU et GPU, et à travers différents paradigmes architecturaux (modèle isolé, Retrieval-Augmented Generation [RAG] et systèmes Multi-Agents), met en exergue plusieurs tendances scientifiques majeures qui redéfinissent les bonnes pratiques pour le déploiement d'assistants locaux.

1. Primauté de la spécialisation du domaine sur la scalabilité des paramètres : Les résultats démontrent de manière empirique que l'entraînement spécifique (fine-tuning) sur des tâches Text-to-SQL constitue le facteur déterminant de la performance. Le modèle `XiYanSQL-QwenCoder-7B` s'impose systématiquement comme la référence de cette étude avec un EM moyen de **42,12 %** et un EX moyen de **72,29 %** sur l'ensemble des six scénarios, dominant les deux autres modèles qui plafonnent tous deux à **57,71 %** d'EX moyen. Cette supériorité valide l'hypothèse selon laquelle l'acquisition de représentations sémantiques spécifiques aux schémas relationnels et à la logique SQL est plus critique que la capacité de raisonnement généraliste.

2. Impact différentiel du mécanisme de Retrieval-Augmented Generation (RAG) : L'intégration d'un module de récupération documentaire s'avère particulièrement bénéfique pour les modèles compacts ou moins spécialisés. Granite-4.0-Tiny gagne **2,50 points** d'EX par rapport au CPU (68,25 % vs 65,50 %), et SLM-SQL-1.5B **2 points** (71,50 % vs 69,50 %). L'effet le plus spectaculaire du RAG est sur les requêtes complexes : XiYanSQL atteint **100 %** d'EX sur le niveau Complexe, un résultat inégalé dans toute l'étude. En revanche, le gain global pour XiYanSQL reste modéré (83,25 % vs 80,25 %), confirmant que ce modèle a déjà internalisé une représentation robuste des structures relationnelles.

3. Limites de l'approche Multi-Agent dans la génération SQL : Contrairement aux attentes théoriques, l'architecture Multi-Agent dégrade les performances globales par rapport au RAG pour tous les modèles (XiYanSQL : 73,75 % vs 83,25 % ; SLM-SQL-1.5B : 66,00 % vs 71,50 % ; Granite : 60,50 % vs 68,25 %). Elle apporte cependant un avantage sur les requêtes complexes, où elle surpasse le RAG pour SLM-SQL-1.5B (**50 %** vs 25 %) et reste équivalente pour XiYanSQL (**75 %** vs 100 % RAG). Ce phénomène s'explique par la propagation d'erreurs en cascade et la perte de cohérence sémantique lors des échanges intermédiaires entre agents.

4. Évaluation critique de l'accélération matérielle GPU : L'accélération GPU bénéficie principalement à SLM-SQL-1.5B (gain de **3,3**× sur le temps d'inférence : 3 525 ms → 1 076 ms) et à XiYanSQL-QwenCoder-7B (gain de **2,4**× : 9 252 ms → 3 916 ms). En revanche, Granite-4.0-Tiny n'en bénéficie pas, avec un temps qui passe de 8 338 ms à 9 287 ms sur GPU. Cette asymétrie suggère une incompatibilité architecturale entre ce modèle et la plateforme GPU utilisée, et constitue une limite importante pour son déploiement en production.

5. Défis de la généralisation multilingue et biais des corpus d'entraînement : La comparaison inter-langues met en évidence une dégradation systématique des métriques pour l'ensemble des modèles sur le corpus arabe. Les scores EX sur les requêtes simples chutent de l'ordre de **14 points** pour XiYanSQL (87,90 % → 73,89 %), de **20 points** pour SLM-SQL-1.5B (77,71 % → 57,01 %) et de **13 points** pour Granite (72,93 % → 59,24 %). Ce décrochage s'explique par le biais inhérent aux jeux de données d'entraînement de référence (Spider, BIRD, WikiSQL), massivement dominés par la langue anglaise.

6. Persistance des verrous scientifiques liés à la complexité des requêtes : L'analyse transversale confirme que la génération de requêtes impliquant des jointures multiples et des sous-requêtes complexes demeure le principal verrou technologique, quel que soit le modèle. En l'absence d'architecture d'aide, le taux d'EX sur les requêtes complexes est **0 %** pour tous les modèles (CPU et GPU). Seul le RAG permet de débloquer ce niveau avec un résultat exceptionnel de **100 %** pour XiYanSQL, tandis que le Multi-Agent offre une alternative plus équilibrée (**75 %** pour XiYanSQL, **50 %** pour SLM-SQL-1.5B, **25 %** pour Granite). Cette contrainte justifie l'orientation des recherches futures vers des paradigmes hybrides combinant RAG, Chain-of-Thought (CoT) et validation syntaxique en boucle fermée.

7. Évaluation des approches basées sur Pandas et Pandas-AI pour l'interrogation de données tabulaires : L'extension de l'évaluation aux approches basées sur la génération de code Python via Pandas apporte des enseignements complémentaires. XiYanSQL-QwenCoder-7B conserve sa supériorité avec **49,75 %** d'EX global, devançant Granite (**35,25 %**) et SLM-SQL-1.5B (**20,75 %**). Cependant, l'EM est nul pour les trois modèles dans ce scénario, confirmant que le paradigme Pandas est fondamentalement différent du SQL. Sur les requêtes simples, XiYanSQL atteint **65,22 %** d'EX, ce qui reste nettement inférieur à ses performances SQL classiques (87,90 % en CPU). Par ailleurs, l'approche Pandas-AI + SLM n'a produit aucun résultat exploitable dans les conditions expérimentales retenues, l'ensemble des tests s'étant soldés par des dépassements de temps d'exécution (*timeouts*), y compris sur plateformes GPU.

Conclusion de la discussion

En somme, cette étude établit que l'optimisation des assistants Text-to-SQL locaux ne repose pas sur une course à la taille des modèles (*scaling law*), mais sur une spécialisation ciblée des données d'entraînement, une gestion rigoureuse du contexte et une architecture adaptée aux contraintes de robustesse logique. `XiYanSQL-QwenCoder-7B` incarne cet équilibre optimal avec un EX moyen de **72,29 %** sur six scénarios, tandis que **SLM-SQL-1.5B** conserve une niche d'utilité pour les environnements à ressources extrêmement contraintes, notamment grâce à son temps d'inférence CPU de seulement **3 525 ms** et sa stabilité remarquable (σ = 2 726 ms).

Chapitre 5

Conception et développement du système ASK-DATA

5.1 Introduction

Ce chapitre présente la conception et le développement d'ASK-DATA, une application intelligente de type **Text-to-SQL** déployée entièrement en local. À partir des modèles SLM sélectionnés au chapitre précédent, nous décrivons ici leur intégration dans un système fonctionnel, opérationnel sur des données réelles.

Nous commençons par une présentation générale du système, ses objectifs et ses cas d'utilisation, puis nous détaillons l'analyse des besoins, la conception logicielle et la modélisation UML. Les choix d'implémentation sont justifiés — notamment le recours à **Streamlit** pour l'interface web et à **LM Studio** pour l'inférence — avant d'aborder les contraintes de sécurité et de confidentialité qui constituent un pilier fondamental du projet.

5.2 Présentation générale d'ASK-DATA

5.2.1 Objectifs fonctionnels

ASK-DATA est un système **Text-to-SQL** local dont la vocation première est de rendre l'interrogation de bases de données accessible à tout utilisateur, quelle que soit son expertise technique. Le modèle SLM sous-jacent est exécuté via **LM Studio**, sans aucune dépendance à un service cloud, ce qui garantit la confidentialité des données traitées.

Les objectifs métier qui ont guidé la conception du système sont les suivants :

- **Démocratiser l'accès aux données** : supprimer la barrière du langage SQL en permettant à tout utilisateur, même non technique, d'interroger ses données en langage naturel (français ou anglais), sans formation préalable.
- **Garantir la souveraineté et la confidentialité des données** : traiter l'intégralité des données en local, sur la machine de l'utilisateur, sans aucune transmission vers un service cloud ou un réseau externe, afin de répondre aux exigences de confidentialité des entreprises et institutions.
- **Offrir une expérience utilisateur guidée et autonome** : accompagner l'utilisateur pas à pas, de l'import de son fichier jusqu'à l'export des résultats, via une interface structurée en étapes claires, sans nécessiter l'intervention d'un expert informatique.
- **Permettre la réutilisation et le partage des résultats** : rendre les résultats d'analyse directement exploitables, sous forme de graphiques interactifs et de rapports PDF exportables, pour faciliter la prise de décision et le partage en équipe.

5.2.2 Cas d'utilisation

Le système ASK-DATA cible deux profils d'utilisateurs principaux :

- **Utilisateur métier** : charge un fichier CSV ou Excel contenant ses données, pose des questions en langage naturel (ex. : *« Combien de chanteurs avons-nous ? »*), visualise et exporte les résultats sans écrire une seule ligne de SQL.
- **Utilisateur technique** : charge une base SQLite, consulte le SQL généré, inspecte le prompt complet et les données exemples envoyés au LLM, et évalue la qualité de la génération.

Le flux d'utilisation suit trois étapes guidées par un stepper visuel intégré dans l'interface :

1. **Étape 1 — Chargement** : l'utilisateur dépose un fichier (CSV, Excel ou SQLite) via la barre latérale. Le système charge les données en mémoire via SQLite in-memory pour les fichiers tabulaires (voir Annexe B, Figure B.1).

2. **Étape 2 — Interrogation** : l'utilisateur saisit sa question en langage naturel. Le système extrait le schéma et les données exemples, construit le prompt et génère le SQL via le LLM local (voir Annexe B, Figure B.2).

3. **Étape 3 — Analyse & Résultats** : le système affiche le tableau de résultats, les métriques de performance (temps de génération, temps total, nombre de lignes), les graphiques interactifs et le bouton d'export PDF (voir Annexe B, Figures B.3 et B.4).

5.2.3 Architecture générale

ASK-DATA repose sur une architecture locale à trois couches :

- **Couche interface** : application web locale développée avec **Streamlit**, accessible via navigateur sur `localhost:8501`.
- **Couche traitement** : pipeline Text-to-SQL orchestrant l'extraction du schéma, la construction du prompt, l'appel au LLM et l'exécution SQL.
- **Couche modèle** : serveur LLM local **LM Studio** exposant une API compatible OpenAI sur `localhost:1234`, chargeant les modèles SLM au format GGUF.

5.3 Analyse des besoins

5.3.1 Besoins fonctionnels

1. **Chargement multi-sources** : le système doit accepter trois types de sources de données — fichiers CSV (`.csv`), fichiers Excel (`.xlsx`/`.xls`) et bases SQLite (`.db`/`.sqlite`) — chargés via un composant d'upload dans la barre latérale.

2. **Extraction automatique du contexte** : le système doit extraire automatiquement le schéma DDL de toutes les tables (noms, colonnes, types) ainsi que 2 lignes de données réelles par table, et les intégrer dans le prompt envoyé au LLM.

3. **Génération SQL par LLM local** : le système doit générer une requête SQL syntaxiquement valide à partir d'une question en langage naturel, en appelant le modèle SLM via l'API LM Studio sur `localhost:1234`, sans recourir à aucun service externe.

4. **Exécution et affichage des résultats** : le système doit exécuter la requête SQL générée sur la base SQLite et afficher les résultats sous forme de tableau interactif (`st.dataframe`), avec gestion et affichage des erreurs d'exécution sans bloquer l'application.

5. **Visualisation graphique interactive** : le système doit proposer quatre types de graphiques interactifs via Plotly — barres, camembert, lignes et nuage de points — avec sélection libre des axes X et Y parmi toutes les colonnes de la table source.

6. **Export PDF** : le système doit permettre l'export des résultats en fichier PDF téléchargeable, incluant la question posée, le schéma de la base de données et le tableau de résultats complet.

7. **Transparence du pipeline** : le système doit afficher, via des expandeurs dédiés, le SQL généré, le prompt complet envoyé au LLM, les données exemples utilisées comme contexte, ainsi que les métriques de performance (temps de génération SQL et temps total en millisecondes).

8. **Réinitialisation de session** : le système doit permettre la réinitialisation complète de la session via un bouton dédié, fermant la connexion SQLite, vidant l'état de l'application et réinitialisant le composant d'upload.

5.3.2 Besoins non fonctionnels

- **Confidentialité** : aucune donnée ne doit quitter l'environnement local. Toutes les inférences sont réalisées en local via LM Studio.

- **Performance** : le temps de génération SQL doit rester acceptable pour une utilisation interactive, mesuré et affiché à l'utilisateur.

- **Compatibilité** : le système doit fonctionner sans GPU dédié, en mode CPU uniquement si nécessaire.

- **Robustesse** : les erreurs SQL (syntaxe, exécution) doivent être interceptées et affichées sans bloquer l'application.

- **Modularité** : le modèle SLM utilisé doit être facilement interchangeable via la configuration de LM Studio, sans modifier le code source.

- **Utilisabilité** : l'interface doit guider l'utilisateur via un stepper visuel à trois étapes, sans nécessiter de connaissances techniques.

5.4 Conception du système

5.4.1 Architecture logicielle

L'architecture logicielle d'ASK-DATA suit un modèle en couches séparées :

- **Couche présentation** : `Streamlit` gère l'interface utilisateur, le stepper, les formulaires de saisie, les tableaux de résultats et les graphiques Plotly.
- **Couche métier** : les fonctions `get_schema_string()`, `get_all_tables_sample_data()`, `generate_sql()` et `execute_sql()` constituent le pipeline de traitement Text-to-SQL.
- **Couche données** : `SQLite` (in-memory pour CSV/Excel, fichier pour les bases natives) assure la persistance et l'exécution des requêtes.
- **Couche inférence** : `LM Studio` expose une API compatible OpenAI sur `localhost:1234`, permettant l'appel au modèle SLM via le client `openai.OpenAI`.

5.4.2 Architecture applicative

Le pipeline applicatif d'ASK-DATA suit les étapes suivantes pour chaque requête :

1. **Chargement des données** : le fichier déposé est lu via `pandas` — `pd.read_csv()` pour les fichiers CSV, `pd.read_excel()` pour les fichiers Excel — ou connecté directement pour les fichiers SQLite. Pour les fichiers tabulaires, les données sont chargées dans une base SQLite in-memory via `df.to_sql()`, avec normalisation du nom de table (`re.sub(r'^a-zA-Z0-9_]', '_', table_name)`).
2. **Extraction du contexte** : `get_schema_string()` interroge `sqlite_master` et extrait le schéma DDL de toutes les tables sous la forme `TABLE nom(col1 TYPE1, col2 TYPE2, ...)`. La fonction `get_all_tables_sample_data()` récupère 2 lignes réelles par table via `pd.read_sql()`, formatées en Markdown pour enrichir le prompt.
3. **Génération SQL** : `generate_sql()` construit le prompt système (*rôle expert SQL, SQL uniquement, sans explication*) et le prompt utilisateur (schéma + données exemples + question) et les envoie au LLM local via l'API LM Studio. Le SQL brut retourné est nettoyé par suppression des balises Markdown (```` ```sql ```` et ```` ``` ````) via `re.sub()`.
4. **Exécution SQL** : `execute_sql()` exécute la requête via `pd.read_sql_query()` et retourne un tuple `(DataFrame, erreur)`. En cas d'erreur SQL, le message est affiché à l'utilisateur sans bloquer l'application.

5. **Visualisation graphique** : la fonction `render_chart()` propose quatre types de graphiques interactifs via **Plotly** : barres (`go.Bar`), camembert (`go.Pie`), lignes (`go.Scatter` mode `lines+markers`) et nuage de points (`go.Scatter` mode `markers`). L'utilisateur choisit les axes X et Y parmi toutes les colonnes de la table source, chargées via `get_table_columns()`.

6. **Export PDF** : la fonction `df_to_pdf_bytes()` génère un PDF incluant la question posée, le schéma de la base (tronqué à 300 caractères) et le tableau de résultats complet, via `fpdf.FPDF`. Le fichier est proposé en téléchargement via `st.download_button()`.

7. **Affichage final** : les résultats sont affichés avec trois métriques de performance — nombre de lignes retournées, temps de génération SQL et temps total en millisecondes — calculées via `time.perf_counter()`, ainsi que des expandeurs pour inspecter le SQL généré et le prompt complet envoyé au LLM.

5.4.3 Architecture de déploiement

ASK-DATA est déployé entièrement en local sur la machine d'un seul utilisateur. Les composants suivants doivent être actifs simultanément :

- **LM Studio** : serveur local exposant le modèle SLM sur `http://localhost:1234/v1`, avec le modèle GGUF chargé.
- **Streamlit** : serveur web local lancé via `streamlit run askdata_app.py`, accessible sur `http://localhost:8501`.
- **Navigateur web** : interface utilisateur accessible depuis tout navigateur moderne sur la même machine.

5.5 Modélisation UML du système ASK-DATA

5.5.1 Diagramme de cas d'utilisation

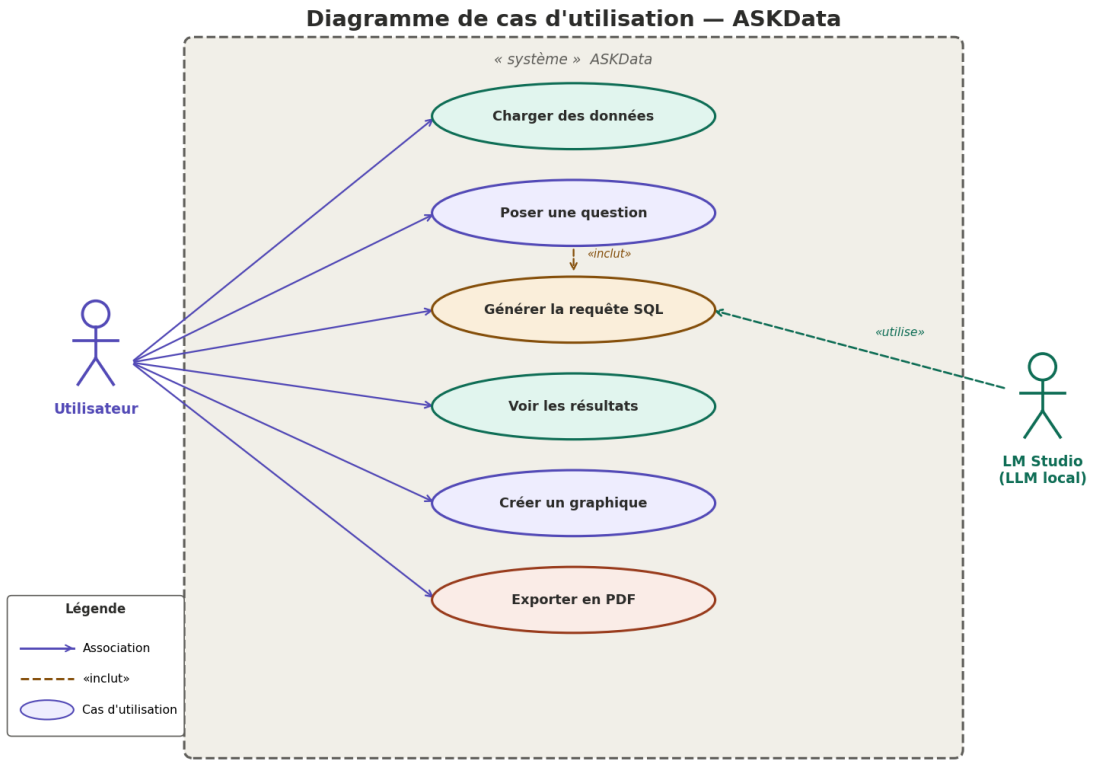


FIG. 5.1 : Diagramme de cas d'utilisation du système ASK-DATA

La figure 5.1 présente le diagramme de cas d'utilisation du système ASK-DATA. Deux acteurs y sont représentés : **l'Utilisateur** (à gauche), acteur principal humain, et **LM Studio (LLM local)** (à droite), acteur secondaire système. L'Utilisateur interagit directement avec cinq cas d'utilisation : *Charger des données*, *Poser une question*, *Voir les résultats*, *Créer un graphique* et *Exporter en PDF*. Le cas *Poser une question* inclut (`«inclut»`) le cas *Générer la requête SQL*, ce qui signifie que toute interrogation déclenche automatiquement la génération SQL. Ce dernier cas est alimenté par l'acteur **LM Studio** via une relation `«utilise»`, matérialisant l'appel au modèle de langage local pour produire la requête SQL correspondant à la question posée.

5.5.2 Diagramme de classes

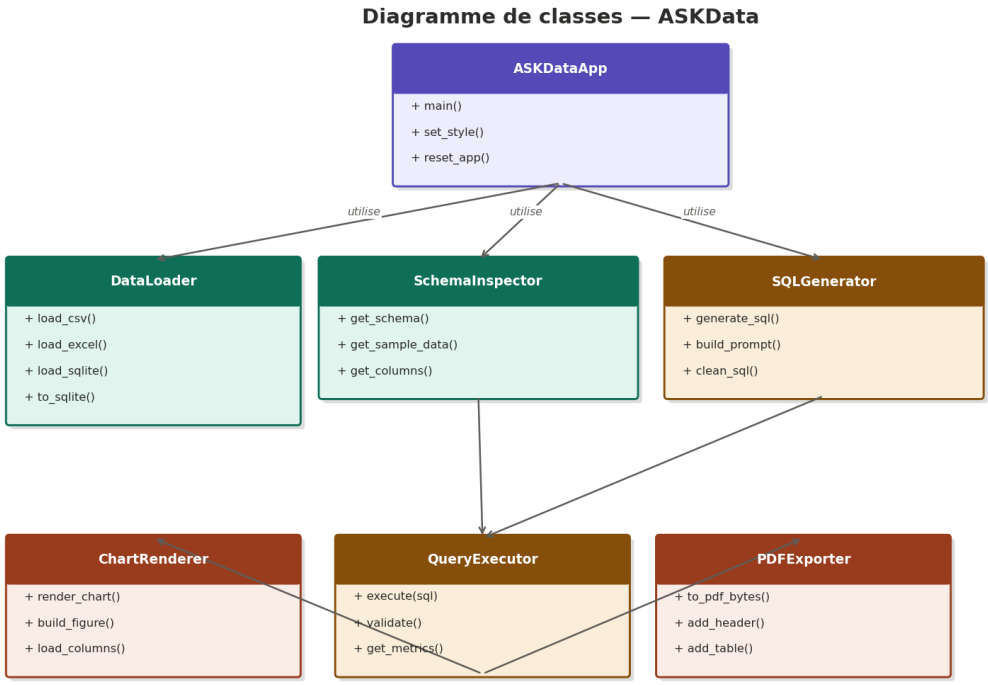


FIG. 5.2 : Diagramme de classes du système ASK-DATA

La figure 5.2 illustre le diagramme de classes du système ASK-DATA. La classe centrale **ASKDataApp** constitue le point d'entrée de l'application, elle expose trois méthodes publiques : `main()`, `set_style()` et `reset_app()`. Elle *utilise* trois classes de traitement : **DataLoader**, **SchemaInspector** et **SQLGenerator**. La classe **DataLoader** prend en charge le chargement des sources de données via les méthodes `load_csv()`, `load_excel()`, `load_sqlite()` et `to_sqlite()`. La classe **SchemaInspector** extrait le contexte de la base de données grâce aux méthodes `get_schema()`, `get_sample_data()` et `get_columns()`. La classe **SQLGenerator** assure la génération de la requête SQL via `generate_sql()`, `build_prompt()` et `clean_sql()`. En aval, **SchemaInspector** alimente la classe **QueryExecutor** (`execute()`, `validate()`, `get_metrics()`), qui retourne les résultats exploités par les deux classes de sortie : **ChartRenderer** (`render_chart()`, `build_figure()`, `load_columns()`) pour la visualisation graphique, et **PDFExporter** (`to_pdf_bytes()`, `add_header()`, `add_table()`) pour la génération du rapport PDF.

5.5.3 Diagrammes de séquence

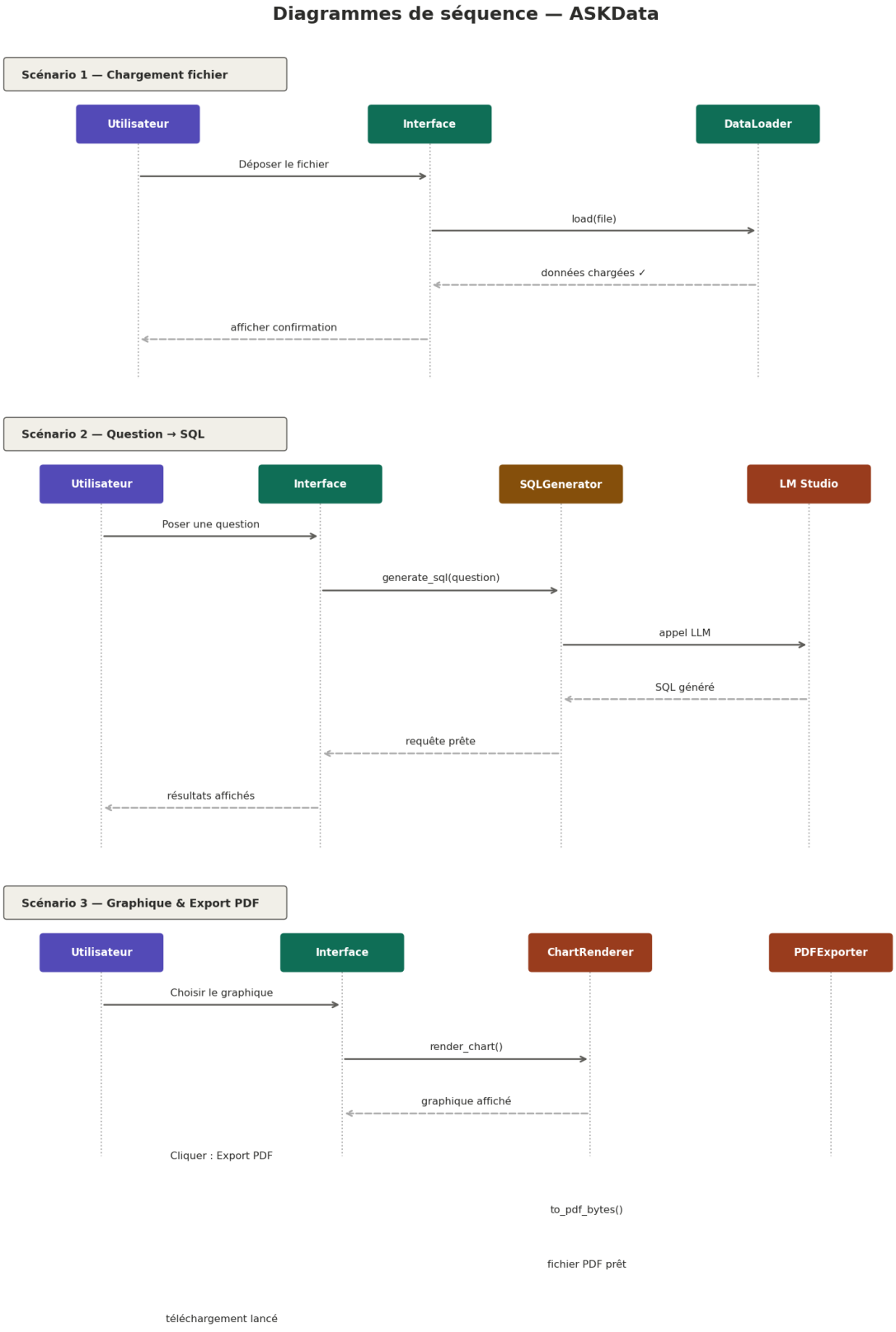


FIG. 5.3 : Diagrammes de séquence — trois scénarios principaux

La figure 5.3 présente les diagrammes de séquence couvrant les trois scénarios principaux du système :

- **Scénario 1 — Chargement fichier** : ce scénario implique trois participants — *Utilisateur*, *Interface* et *DataLoader*. L'utilisateur dépose le fichier sur l'interface, qui transmet un appel `load(file)` au `DataLoader`. Celui-ci retourne la confirmation *données chargées* à l'interface, qui affiche à son tour la confirmation à l'utilisateur.
- **Scénario 2 — Question → SQL** : ce scénario implique quatre participants — *Utilisateur*, *Interface*, *SQLGenerator* et *LM Studio*. L'utilisateur pose une question : l'interface appelle `generate_sql(question)` sur le `SQLGenerator`, qui effectue un appel LLM vers **LM Studio**. LM Studio retourne le SQL généré au `SQLGenerator`, qui transmet *requête prête* à l'interface. L'interface affiche enfin les résultats à l'utilisateur.
- **Scénario 3 — Graphique & Export PDF** : ce scénario implique quatre participants — *Utilisateur*, *Interface*, *ChartRenderer* et *PDFExporter*. L'utilisateur choisit un type de graphique : l'interface appelle `render_chart()` sur le `ChartRenderer`, qui retourne le graphique affiché. L'utilisateur clique ensuite sur *Export PDF* : l'interface appelle `to_pdf_bytes()` sur le `PDFExporter`, qui signale *fichier PDF prêt*, déclenchant le téléchargement du fichier.

5.6 Implémentation

5.6.1 Technologies utilisées

Tab. 5.1 : Technologies utilisées dans ASK-DATA.

Composant	Technologie	Rôle
Interface web	Streamlit	UI web locale interactive
LLM local	LM Studio	Serveur d'inférence SLM (API OpenAI)
Manipulation données	Pandas	Chargement CSV/Excel, DataFrames
Base de données	SQLite	Stockage et exécution SQL
Visualisation	Plotly	Graphiques interactifs
Export PDF	fpdf	Génération de rapports PDF
Client LLM	openai (Python)	Appels API vers LM Studio
Langage	Python 3.10+	Développement backend & frontend

5.6.2 Backend

Le backend d'ASK-DATA est entièrement développé en Python. Il comprend :

- **Extraction du schéma** : la fonction `get_schema_string()` interroge `sqlite_master` pour extraire les tables et colonnes, et les formate sous la forme `TABLE nom(col TYPE, ...)`. La fonction `get_all_tables_sample_data()` lit 2 lignes réelles par table via `pd.read_sql()`, formatées en Markdown pour le prompt via `to_markdown()`.
- **Génération SQL** : la fonction `generate_sql()` construit le prompt (system + user) et appelle le modèle SLM via `openai.OpenAI(base_url='http://localhost:1234/v1', api_key='lm-studio')`. Le client est mis en cache via `@st.cache_resource` pour éviter des reconnexions inutiles. Le SQL retourné est nettoyé des balises Markdown (```` ```sql ```` et ```` ``` ````) via `re.sub()`.
- **Exécution SQL** : la fonction `execute_sql()` exécute la requête via `pd.read_sql_query()` et retourne un tuple `(DataFrame, erreur)`. Les exceptions sont capturées et retournées proprement, sans bloquer l'application.
- **Colonnes de table** : la fonction `get_table_columns()` interroge `PRAGMA table_info()` pour retourner la liste complète des colonnes d'une table, utilisée par `render_chart()` pour proposer tous les axes possibles de visualisation.
- **Export PDF** : la fonction `df_to_pdf_bytes()` génère un PDF via `fpdf.FPDF`, incluant le titre, la question posée (tronquée à 200 caractères), le schéma de la base (tronqué à 300 caractères) et le tableau de résultats complet avec en-têtes.

5.6.3 Frontend

L'interface utilisateur est développée avec **Streamlit** et enrichie d'un style CSS personnalisé injecté via `st.markdown(..., unsafe_allow_html=True)`. Elle comprend :

- **Barre latérale** : sélecteur de source de données (radio button entre *CSV / Excel* et *SQLite*), uploader de fichiers avec clé dynamique (`upload_key`) pour permettre le remplacement de fichier, et bouton de réinitialisation (`reset_app()`) qui ferme la connexion SQLite, vide le `session_state` et incrémente la clé d'upload.
- **Header dynamique** : indicateur de statut de connexion — point vert (`#4ade80`) si une base est connectée, jaune (`#fcd34d`) en attente — avec texte *CONNECTÉ* ou *EN ATTENTE*.
- **Stepper visuel** : guide l'utilisateur à travers les 3 étapes — *Chargement* (étape 0), *Interrogation* (étape 1) et *Analyse & Résultats* (étape 2) — avec états visuels distincts :

complété (vert), actif (violet animé) et à venir (numéro grisé) (voir Annexe B, Figures B.1, B.2 et B.3).

- **Zone de question** : formulaire Streamlit (`st.form`) avec zone de texte et bouton de soumission *Analyser & Exécuter*.
- **Zone de résultats** : métriques (lignes retournées, temps de génération SQL, temps total), tableau interactif (`st.dataframe`), expandeurs pour le SQL généré et le contexte complet (question, schéma, données exemples, prompt complet), graphiques Plotly et bouton d'export PDF.

5.6.4 Intégration du modèle retenu

Le modèle SLM retenu suite aux expérimentations du chapitre précédent est chargé dans **LM Studio** au format GGUF. L'intégration est réalisée via le client Python `openai` configuré pour pointer vers le serveur local :

```
API_BASE   = "http://localhost:1234/v1"
API_KEY    = "lm-studio"
MODEL_NAME = "local-model"
```

Le client est instancié une seule fois grâce au décorateur `@st.cache_resource` appliqué à la fonction `get_openai_client()`, ce qui évite de recréer la connexion à chaque requête. Le modèle est interchangeable sans modifier le code : il suffit de charger un autre fichier GGUF dans LM Studio et de mettre à jour `MODEL_NAME`. Les paramètres de génération sont fixés pour garantir un comportement déterministe : température à `0.0` et nombre maximum de tokens à `2048`.

5.6.5 Pipeline Text-to-SQL

Le pipeline complet Text-to-SQL d'ASK-DATA suit les étapes suivantes pour chaque requête utilisateur :

1. **Extraction du contexte** : schéma DDL extrait via `get_schema_string()` + 2 lignes d'exemples par table via `get_all_tables_sample_data()`.
2. **Construction du prompt** : system prompt définissant le rôle d'expert SQL (génération de SQL uniquement, sans explication ni Markdown) + user prompt assemblant le schéma, les données exemples et la question de l'utilisateur.
3. **Appel LLM** : envoi du prompt à LM Studio via `client.chat.completions.create()`, réception du SQL brut avec mesure du temps de génération via `time.perf_counter()`.
4. **Nettoyage** : suppression des balises Markdown (```` ```sql ```` et ```` ``` ````) via deux appels `re.sub()`.

5. **Exécution** : exécution du SQL nettoyé sur SQLite via `pd.read_sql_query()`, retour d'un DataFrame pandas ou d'un message d'erreur.

6. **Visualisation graphique** : appel à `render_chart()` avec le DataFrame résultat, la connexion SQLite et la table source sélectionnée, génération du graphique choisi via Plotly (`go.Bar`, `go.Pie`, `go.Scatter`).

7. **Affichage & Export** : affichage du tableau interactif, des métriques de performance, du SQL généré et du prompt complet dans des expandeurs, export PDF disponible via `df_to_pdf_bytes()` transmettant la question, le schéma et le tableau.

5.7 Sécurité et confidentialité

5.7.1 Déploiement local

L'ensemble du système ASK-DATA est déployé et exécuté localement sur la machine de l'utilisateur. Aucune donnée (questions, schémas, résultats) ne transite par un service externe ou un réseau public. Le serveur LM Studio est accessible uniquement sur `localhost:1234` et l'interface Streamlit sur `localhost:8501`, garantissant un isolement total du système.

5.7.2 Protection des données

- **Données en mémoire** : les fichiers CSV et Excel sont chargés dans une base SQLite *in-memory* via `sqlite3.connect(":memory:", check_same_thread=False)`, sans aucune écriture sur le disque. Les données sont effacées à la fermeture de la session ou lors de la réinitialisation via `reset_app()`, qui appelle `conn.close()` avant de vider le `session_state`.

- **Fichiers temporaires** : les bases SQLite uploadées sont écrites dans un répertoire temporaire système via `tempfile.NamedTemporaryFile(delete=False, suffix=".db")`, et non dans un répertoire accessible publiquement.

- **Aucune journalisation externe** : le système ne journalise aucune donnée utilisateur vers des services tiers. Le module `warnings` est configuré pour supprimer les avertissements système (`warnings.filterwarnings("ignore")`).

5.7.3 Gestion des accès

ASK-DATA est une application locale mono-utilisateur. La gestion des accès repose sur l'isolation au niveau système d'exploitation : l'accès à l'interface est strictement limité à la machine hôte via `localhost`, sans possibilité de connexion depuis un réseau externe. Dans

une perspective d'évolution, une couche d'authentification et de gestion des sessions pourrait être ajoutée pour supporter un déploiement multi-utilisateurs.

5.8 Conclusion

Ce chapitre a présenté la conception et le développement complet du système ASK-DATA. Nous avons décrit l'architecture locale à trois couches (interface, traitement, inférence), les choix technologiques (Streamlit, LM Studio, SQLite, Plotly, fpdf), et le pipeline Text-to-SQL implémenté. La modélisation UML a permis de formaliser les interactions entre les composants du système.

ASK-DATA répond aux contraintes fondamentales du projet : déploiement entièrement local, confidentialité totale des données, interface utilisateur accessible aux non-techniciens, et intégration transparente des modèles SLM sélectionnés lors des expérimentations. La conclusion générale qui suit synthétisera les apports de ce travail et ouvrira sur les perspectives futures.

Conclusion Générale

Ce mémoire s'est inscrit dans le contexte de la **démocratisation de l'accès aux données** et de la nécessité de développer des **systèmes intelligents** capables de faciliter l'interaction entre les utilisateurs non spécialistes et les bases de données relationnelles. Face aux limitations des grands modèles de langage (LLM), notamment en termes de **coûts computationnels**, de **confidentialité des données** et de dépendance aux infrastructures cloud, les **Small Language Models (SLM)** apparaissent comme une alternative particulièrement prometteuse pour le développement d'**assistants locaux** dédiés aux tâches de traitement du langage naturel appliquées aux bases de données.

L'objectif principal de cette recherche était d'identifier **l'architecture la plus adaptée** au développement du système **ASK-DATA** à travers une **étude comparative approfondie** de plusieurs approches d'exploitation des SLM : l'utilisation directe des modèles, l'intégration d'un mécanisme **Retrieval-Augmented Generation (RAG)**, l'adoption d'une architecture **Multi-Agent** et l'exploration d'une approche alternative fondée sur la **génération de code Python** à l'aide de Pandas et Pandas-AI.

Afin d'atteindre cet objectif, une **campagne expérimentale exhaustive** a été menée sur le **benchmark Spider**, en considérant **dix Small Language Models** représentatifs de différentes familles de modèles open source et en évaluant leurs performances sur des plateformes **CPU et GPU** selon plusieurs critères complémentaires : qualité de génération, **exactitude d'exécution**, robustesse syntaxique, **temps d'inférence**, consommation des ressources matérielles et aptitude au déploiement local.

Les résultats obtenus mettent en évidence plusieurs **contributions scientifiques majeures**. Premièrement, ils démontrent que la **spécialisation des modèles** sur le domaine **Text-to-SQL** constitue un facteur de performance plus déterminant que l'augmentation du nombre de paramètres ou de la taille de la fenêtre de contexte. Les modèles spécifiquement entraînés sur des tâches SQL présentent systématiquement de meilleures performances que les modèles généralistes ou ultra-compacts.

Deuxièmement, l'étude montre que le mécanisme **RAG peut améliorer significativement** les performances des modèles compacts ou insuffisamment spécialisés en enrichissant leur compréhension du schéma de la base de données. Toutefois, son apport devient **marginal pour les modèles hautement spécialisés**, qui disposent déjà de représentations internes suffisamment riches pour traiter efficacement les requêtes Text-to-SQL.

Troisièmement, les résultats **remettent en question l'intérêt des architectures Multi-Agent** pour la génération de requêtes SQL. Malgré leurs avantages théoriques en matière de décomposition des tâches et de collaboration entre agents spécialisés, les expérimentations révèlent une **dégradation des performances** due à la propagation des erreurs intermédiaires et à la perte de cohérence sémantique, rendant ces architectures moins adaptées aux contraintes de précision imposées par les systèmes Text-to-SQL.

Quatrièmement, l'évaluation des approches fondées sur la génération de code Python

montre qu'une architecture basée sur **Pandas et les SLM** constitue une **alternative crédible** pour l'interrogation de données tabulaires. En revanche, les expérimentations menées avec **Pandas-AI** ont mis en évidence des **limitations importantes** liées à la surcharge computationnelle induite par ses mécanismes de raisonnement itératif, conduisant à des temps d'exécution incompatibles avec les contraintes d'un déploiement local à ressources limitées.

L'étude met également en évidence la persistance de plusieurs **défis scientifiques majeurs**. Les performances des systèmes évalués **diminuent significativement** face aux requêtes SQL complexes impliquant des jointures multiples, des agrégations imbriquées ou des sous-requêtes avancées. De plus, les expérimentations sur un **corpus arabophone** révèlent un **écart de performance important** par rapport aux corpus anglophones, soulignant les **biais linguistiques** présents dans les jeux de données de référence actuellement utilisés pour l'entraînement et l'évaluation des systèmes Text-to-SQL.

Parmi l'ensemble des modèles étudiés, le modèle `XiYanSQL-QwenCoder-7B` s'impose comme la **solution la plus performante et la plus équilibrée**. Sa robustesse, sa précision, sa capacité de généralisation inter-langues et sa compatibilité avec un déploiement local en font le **candidat le plus approprié** pour la conception du système ASK-DATA. Les résultats obtenus démontrent ainsi qu'un assistant Text-to-SQL local performant ne nécessite pas nécessairement des modèles de très grande taille, mais plutôt une **spécialisation ciblée**, une gestion efficace du contexte et une architecture d'inférence adaptée aux contraintes opérationnelles.

Au-delà des résultats expérimentaux obtenus, ce travail contribue à une **meilleure compréhension des compromis** existants entre performance, coût computationnel et déployabilité des systèmes fondés sur les Small Language Models. Il fournit également un **cadre méthodologique reproductible** pour l'évaluation de futures architectures Text-to-SQL locales.

Enfin, plusieurs **perspectives de recherche** se dégagent de cette étude. Il apparaît particulièrement pertinent d'explorer des **approches hybrides** combinant Retrieval-Augmented Generation, raisonnement explicite de type **Chain-of-Thought** et mécanismes **neuro-symboliques** de validation syntaxique et sémantique. L'enrichissement des **corpus multilingues**, notamment pour la langue arabe, constitue également une voie de recherche prioritaire afin d'améliorer la généralisation des modèles. Enfin, le développement et l'intégration du **prototype complet du système ASK-DATA**, incluant des mécanismes d'explicabilité, de correction automatique des requêtes et de dialogue interactif avec l'utilisateur, représentent une étape naturelle pour la poursuite de ces travaux.

En définitive, cette recherche démontre que les **Small Language Models spécialisés** constituent aujourd'hui une **solution crédible et opérationnelle** pour le développement d'assistants Text-to-SQL locaux, ouvrant ainsi la voie à des systèmes d'accès aux données plus **intelligents**, plus **souverains** et plus **accessibles** à un large éventail d'utilisateurs.

♡

« L'intelligence artificielle locale n'est pas une contrainte — c'est une opportunité de reprendre le contrôle de nos données. »

Annexes

Annexe A : Exemples de requêtes SQL

A.1 Requêtes simples

Exemple 1 : How many singers do we have ?

```
SELECT count(*) FROM singer;
```

Exemple 2 : What are the names of all singers ?

```
SELECT name FROM singer;
```

A.2 Requêtes avec filtrage

Exemple 3 : Singers between 20 and 30 years old ?

```
SELECT name FROM singer WHERE age BETWEEN 20 AND 30;
```

A.3 Requêtes d'agrégation

Exemple 4 : Countries that have more than 2 singers.

```
SELECT country, count(*) FROM singer GROUP BY country HAVING count(*)
> 2;
```

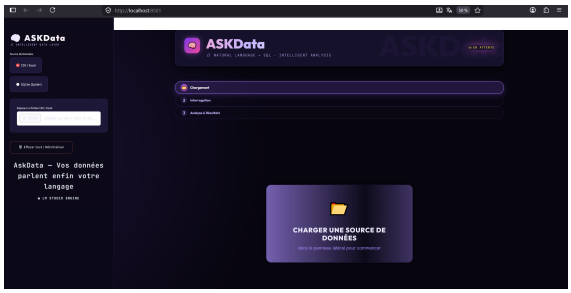
A.4 Requêtes complexes

Exemple 5 : Name of the singer with the highest age ?

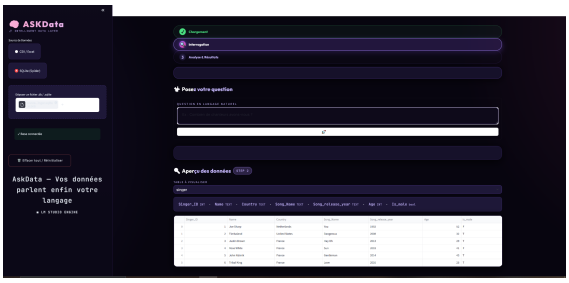
```
SELECT name FROM singer WHERE age = (SELECT max(age) FROM singer);
```

Annexe B : Captures d'écran du système ASK-DATA

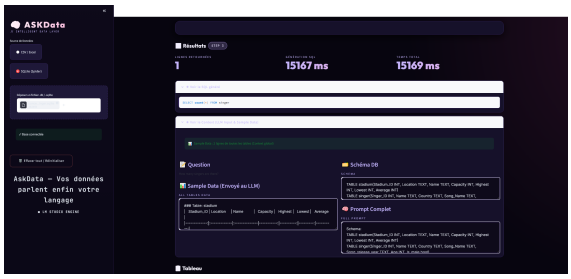
Cette annexe présente les principales interfaces du système ASK-DATA.



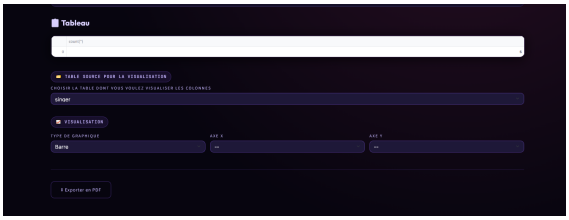
B.1 : Chargement



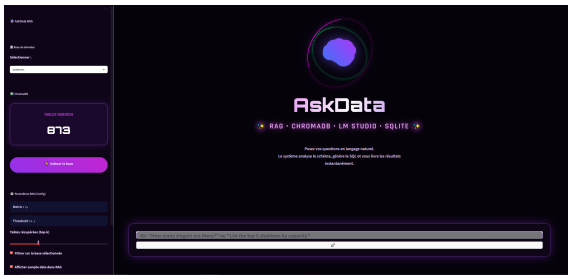
B.2 : Interrogation



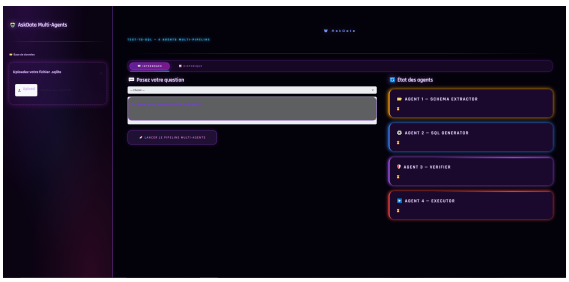
B.3 : Analyse



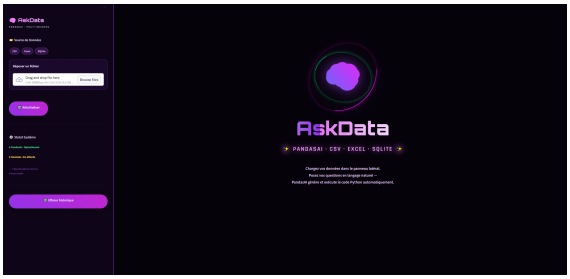
B.4 : Visualisation



B.5 : Interface RAG



B.6 : Multi-Agents



B.7 : Interface Pandas

Annexe C : Guide d'installation (Windows)

C.1 Prérequis

- **RAM :** 16 Go | **GPU :** NVIDIA 4 Go VRAM
- **Logiciels :** Windows 10/11, Python 3.10+, LM Studio 0.4.x

C.2 Installation

1. **LM Studio :** Modèle `XiYanSQL-QwenCoder-7B`.
2. **Environnement :** `python -m venv venv; pip install openai chromadb sqlalchemy sqlglot pandas streamlit`.
3. **Serveur :** Lancer serveur local (`http://localhost:1234/v1`).

C.3 Lancement

Commande : `streamlit run app.py` (Accès : `http://localhost:8501`).

C.4 Résolution des problèmes

Problème	Solution
Connection refused	Démarrer le serveur LM Studio.
CUDA out of memory	Réduire les couches GPU.
Requêtes incorrectes	Tester le mode RAG.
ChromaDB vide	Lancer `index_schemas.py`.

Bibliographie

Anthropic. (2026). Claude Sonnet [Modèle de langage]. https ://www.anthropic.com

Barrere, K. (2023). Architectures de Transformer légères pour la reconnaissance de textes manuscrits anciens [Thèse de doctorat, INSA de Rennes].

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems,* 33, 1877–1901.

Chroma. (2023). ChromaDB (Version 1.5.9) [Logiciel informatique]. *PyPI.* https ://pypi.org/project/chromadb/

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM,* 13(6), 377–387. https ://doi.org/10.1145/362384.362685

Date, C. J. (2003). An introduction to database systems (8e éd.). *Addison-Wesley.*

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT : Pre-training of deep bidirectional transformers for language understanding. *In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies,* Volume 1 (pp. 4171–4186). Association for Computational Linguistics. https ://doi.org/10.18653/v1/N19-1423

Duan, Z., & Wang, J. (2024, novembre). Revealing the powerful ability of large language model prompts in text-to-SQL tasks. *In International Conference on Algorithms, High Performance Computing, and Artificial Intelligence* (AHPCAI 2024) (Vol. 13403, pp. 417–422). SPIE.

Efimov, A. I. (2025). Localizing invariants of inverse limits *(arXiv :2502.04123).* https ://arxiv.org/abs/2502.04123

Elmasri, R., & Navathe, S. B. (2015). Fundamentals of database systems (7e éd.). *Pearson.*

Galli, G. (2023). PandasAI (Version 3.0.0) [Logiciel informatique]. *PyPI.* https ://pypi.org/project/pandasai/

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2008). Database systems : The complete book (2e éd.). *Pearson Education.*

Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Zhu, Q., ... & He, Y. (2025). DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning *(arXiv :2501.12948).*https ://arxiv.org/abs/2501.12948

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation,* 9(8), 1735–1780. https ://doi.org/10.1162/neco.1997.9.8.1735

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. de las, Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., & Sayed, W. E. (2023). Mistral 7B *(arXiv :2310.06825).*

https ://arxiv.org/abs/2310.06825

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). Scaling laws for neural language models *(arXiv :2001.08361).* https ://arxiv.org/abs/2001.08361

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*, 33, 9459–9474.

Mao, T. (2023). sqlglot (Version 25.34.1) [Logiciel informatique]. *PyPI*. https ://pypi.org/project/sqlglot/

Melton, J., & Simon, A. R. (1993). Understanding the new SQL : A complete guide. *Morgan Kaufmann.*

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space *(arXiv :1301.3781).* https ://arxiv.org/abs/1301.3781

Nouali, S., Badache, I., & Bellot, P. (2025). Exploration du RAG pour la génération de réponses à des questions en contexte éducatif : Étude sur les données SCIQ. *In Actes de l'atelier Intelligence Artificielle générative et ÉDUcation : Enjeux, Défis et Perspectives de Recherche* 2025 (IA-ÉDU) (pp. 29–41).

Sheng, L., & Shuai, X. S. (2025). SLM-SQL : An exploration of small language models for text-to-SQL. *In Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics* (pp. 1497–1512). Association for Computational Linguistics.

Shi, L., Tang, Z., Zhang, N., Zhang, X., & Yang, Z. (2025). A survey on employing large language models for text-to-SQL tasks. *ACM Computing Surveys,* 58(2), 1–37. https ://doi.org/10.1145/3694890

Tham, H. H., Lim, T. M., & Tan, K. S. N. (2025). Comparative survey of small and large language models. *In 2025 IEEE International Conference on Computation, Big-Data and Engineering (ICCBE)* (pp. 329–334). IEEE.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). LLaMA : Open and efficient foundation language models *(arXiv :2302.13971).* https ://arxiv.org/abs/2302.13971

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 5998–6008.

Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Chai, L., Yan, Z., Shi, S., Sun, J., Liu, S., Cheng, F., Pan, C., Yin, M., & Li, Z. (2025). MAC-SQL : A multi-agent collaborative framework for text-

to-SQL. *In Proceedings of the 31st International Conference on Computational Linguistics* (pp. 540–557). Association for Computational Linguistics.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., & Fedus, W. (2022). Emergent abilities of large language models *(arXiv :2206.07682).* https ://arxiv.org/abs/2206.07682

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., & Radev, D. (2018). Spider : A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. *In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (pp. 3911–3921). Association for Computational Linguistics. https ://doi.org/10.18653/v1/D18-1425

Zampieri, N. (2023). Détection des discours haineux dans les réseaux sociaux : *Apport des expressions polylexicales* [Thèse de doctorat, Université de Lorraine].