

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA
RECHERCHE SCIENTIFIQUE

ÉCOLE NATIONALE POLYTECHNIQUE



المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département Génie Industriel

Mémoire de projet de fin d'études

**En vue de l'obtention du diplôme d'Ingénieur d'État en Génie Industriel Option :
Data Science et Intelligence Artificielle**

Conception d'un Outil de BI Conversationnelle Basé sur le RAG
Agentic

KERFA Alia

Sous la direction de **Dr. ARKI Oussama, ENP & Mr. BENATMANE
Mohamed Amine**

Présenté et soutenu publiquement le (20/06/2026)

Composition du jury :

Président : Pr. BELDJOUDI Samia ENP
Promoteur : Dr. ARKI Oussama ENP
Examinatrice : Dr. BARECHE Imene ENP

ENP 2026

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA
RECHERCHE SCIENTIFIQUE

ÉCOLE NATIONALE POLYTECHNIQUE



المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Département Génie Industriel

Mémoire de projet de fin d'études

**En vue de l'obtention du diplôme d'Ingénieur d'État en Génie Industriel Option :
Data Science et Intelligence Artificielle**

Conception d'un Outil de BI Conversationnelle Basé sur le RAG
Agentic

KERFA Alia

Sous la direction de **Dr. ARKI Oussama, ENP & Mr. BENATMANE
Mohamed Amine**

Présenté et soutenu publiquement le (20/06/2026)

Composition du jury :

Président : Pr. BELDJOUDI Samia ENP
Promoteur : Dr. ARKI Oussama ENP
Examinatrice : Dr. BARECHE Imene ENP

ENP 2026

Remerciements

Avant tout, je remercie Allah, le Tout-Puissant, de m'avoir accordé la force, la patience et la persévérance nécessaires pour mener ce travail à son terme.

Je tiens à exprimer ma profonde gratitude à mon encadrant académique, M. Arki Oussama, pour la qualité de son accompagnement, sa disponibilité et ses précieux conseils. Sa rigueur et son professionnalisme ont constitué une source d'inspiration constante tout au long de ce projet.

J'adresse également mes sincères remerciements à M. Benatmane Amine, mon encadrant au sein de l'entreprise Djezzy, pour son soutien, ses orientations pertinentes et la confiance qu'il m'a accordée durant cette expérience.

Un très grand remerciement ainsi qu'une profonde reconnaissance sont adressés à M. Kacimi El Hassani Hichem pour son aide précieuse et son soutien indéfectible tout au long de mon parcours.

Je tiens à remercier chaleureusement Mme Bareche Imene pour avoir accepté d'examiner ce mémoire. Je lui suis reconnaissante pour l'intérêt porté à ce travail, ainsi que pour le temps et l'attention consacrés à son évaluation.

Enfin, j'adresse mes remerciements à l'ensemble du corps professoral de l'École Nationale Polytechnique pour la qualité de l'enseignement dispensé et les connaissances transmises tout au long de ma formation.

Alia Kerfa

الملخص

يقدم هذا البحث تصميم وتطوير نظام ذكي للمساعدة في التحليل واتخاذ القرار، يعتمد على معمارية Agentic RAG متعددة الوكلاء. يتيح النظام للمستخدمين الاستعلام عن مصادر بيانات متغيرة، تشمل قواعد البيانات العلائقية وغير العلائقية، باستخدام اللغة الطبيعية. ويعتمد على آليات الاسترجاع الدلالي للمخططات وعمليات استدلال منظمة وقابلة للتوسع، لتوليد الاستعلامات تلقائياً وإنتاج تحليلات سياقية وإنشاء تمثيلات بيانية مناسبة.

جرى تقييم النظام من خلال دراسة أربعة محاور رئيسية: جودة استرجاع المخططات، ودقة توليد الاستعلامات، وتماسك عملية الاستدلال، ومدى ملاءمة الإجابات المولدة. وقد أظهرت النتائج فعالية نهج RAG Agentic في تطوير أنظمة ذكاء الأعمال الحوارية، من خلال تعزيز المرونة وتحسين القدرة على التكيف والارتقاء بتجربة المستخدم.

الكلمات المفتاحية: الذكاء الاصطناعي، المساعد الذكي، التوليد المعزز بالاسترجاع، النماذج اللغوية الكبيرة، استخراج البيانات، دعم اتخاذ القرار.

Abstract

This thesis presents the design and development of an intelligent analytical assistant system based on a multi-agent Agentic RAG architecture. The system enables users to query heterogeneous data sources using natural language. It incorporates semantic schema retrieval mechanisms and a modular reasoning process to automatically generate queries, produce contextualized analyses, and create adapted visualizations.

The system was evaluated across four dimensions: schema retrieval quality, query generation accuracy, reasoning coherence, and response relevance. The results demonstrate the effectiveness of the Agentic RAG approach for conversational Business Intelligence systems, improving flexibility, adaptability, and overall user experience.

Keywords: Agentic RAG, Artificial Intelligence, Multi-Agent System, Conversational Business Intelligence, Data Analytics.

Résumé

Ce mémoire présente la conception et le développement d'un système intelligent d'assistance analytique basé sur une architecture Agentic RAG multi-agents. Le système permet aux utilisateurs d'interroger des sources de données hétérogènes, relationnelles et non relationnelles, en langage naturel. Il intègre des mécanismes de récupération sémantique des schémas et un processus de raisonnement modulaire afin de générer automatiquement des requêtes, produire des analyses contextualisées et créer des visualisations adaptées.

L'évaluation du système a porté sur la qualité de récupération des schémas, la génération des requêtes, la cohérence du raisonnement et la pertinence des réponses produites. Les résultats obtenus démontrent l'efficacité de l'approche Agentic RAG pour les systèmes de Business Intelligence conversationnelle, en améliorant la flexibilité, l'adaptabilité et l'expérience utilisateur.

Mots-clés : Agentic RAG, Intelligence Artificielle, Système Multi-Agents, Business Intelligence Conversationnelle, Analyse de Données.

Table des matières

Liste des tableaux

Liste des figures

Liste des acronymes

Introduction générale	16
1 État de l’art	19
1.1 Donnée, information et décision	20
1.1.1 Donnée	20
1.1.2 Information	20
1.1.3 Connaissance et décision	20
1.2 Types de données	20
1.2.1 Données structurées	21
1.2.2 Données semi-structurées	21
1.2.3 Données non structurées	21
1.3 Types de bases de données	22
1.3.1 Bases de données relationnelles	22
1.3.2 Bases de données non relationnelles (NoSQL)	23
1.3.2.0.1 Bases clé-valeur.	24
1.3.2.0.2 Bases orientées documents.	24
1.3.2.0.3 Bases orientées colonnes.	24
1.3.2.0.4 Bases de données orientées graphes.	25
1.4 Contexte et définition de la Business Intelligence	25

1.4.1	Contexte	25
1.4.2	Définition de la Business Intelligence	26
1.4.3	Objectifs de la Business Intelligence	26
1.5	Architecture et fonctionnement d'une solution BI	27
1.5.1	Introduction	27
1.5.2	Architecture générale	27
1.5.3	Les sources de données	28
1.5.4	L'entrepôt de données (Data Warehouse)	28
1.5.4.1	Deux approches de conception	28
1.5.4.2	Modèles de conception dimensionnelle	28
1.5.4.2.1	Le schéma en étoile (star schema) :	29
1.5.4.2.2	Le schéma en flocon (snowflake schema) :	29
1.5.4.3	Datamarts	29
1.5.5	Le processus ETL	29
1.5.5.1	Extraction (Extract)	29
1.5.5.2	Transformation (Transform)	29
1.5.5.3	Chargement (Load)	29
1.5.5.4	ETL vs ELT	30
1.5.6	Les outils d'analyse et de visualisation	30
1.5.6.1	Reporting	30
1.5.6.2	OLAP	30
1.5.6.2.1	ROLAP (Relational OLAP) :	30
1.5.6.2.2	MOLAP (Multidimensional OLAP) :	30
1.5.6.2.3	HOLAP (Hybrid OLAP) :	30
1.5.6.3	Tableaux de bord et visualisation	31
1.5.6.4	Analyse avancée	31
1.6	Définition et périmètre	31
1.7	Apprentissage automatique	31
1.7.1	Définition	31
1.7.2	Types d'apprentissage automatique	32

1.8	Apprentissage profond	32
1.8.1	Concepts fondamentaux	32
1.8.2	Architectures de réseaux de neurones profonds	33
1.9	Architecture Transformer	33
1.9.1	Principe général	33
1.9.2	Mécanisme d'attention	34
1.9.3	Importance dans l'IA moderne	35
1.10	Grands modèles de langage (LLMs)	35
1.11	Conclusion	36
2	Du RAG Classique au RAG Agentique : Fondements et Architecture	37
2.1	Architecture générale d'un système RAG	38
2.1.1	Indexation (<i>Indexing</i>)	38
2.1.2	Recherche d'information (<i>Retrieval</i>)	39
2.1.3	Génération (<i>Generation</i>)	39
2.2	Évolution des paradigmes RAG	39
2.2.1	RAG Naïf (<i>Naive RAG</i>)	39
2.2.2	RAG Avancé (Advanced RAG)	40
2.2.3	RAG Modulaire (<i>Modular RAG</i>)	40
2.3	Architecture en modules des systèmes RAG modernes	41
2.3.1	Indexation	41
2.3.2	Pré-retrieval	41
2.3.3	Retrieval	42
2.3.4	Post-retrieval	43
2.3.5	Génération	43
2.4	Raisonnement à logique prédéfinie dans les systèmes RAG	43
2.4.1	Approches basées sur le routage	44
2.4.2	Approches itératives en boucle	45
2.4.3	Approches hiérarchiques en arbre	45
2.4.4	Approches hybrides modulaires	46

2.4.5	Limites structurelles des pipelines RAG prédéfinis	46
2.5	Raisonnement agentique (<i>Agentic Reasoning</i>)	47
2.6	Formalisation générale des systèmes agentiques basés sur les LLM	48
2.6.1	Ensemble des agents (I)	48
2.6.2	État de l'environnement (S)	49
2.6.3	Espace des actions (A)	49
2.6.4	Espace des observations (Ω)	49
2.6.5	Fonction d'observation ($O(o_t^i \mid s_t, h_t^i, Q)$)	49
2.6.6	Politiques de décision (Π)	50
2.6.7	Mémoire des agents (M)	50
2.6.8	Fonction de transition ($\Psi(s_{t+1} \mid s_t, a_t)$)	50
2.7	Propriétés des systèmes <i>Agentic RAG</i>	50
2.8	Taxonomie de l'Agentic RAG	51
2.8.1	Taxonomie architecturale	51
2.8.2	Taxonomie des stratégies de récupération	52
2.8.3	Taxonomie du raisonnement	52
2.8.4	Paradigmes de mémoire et de contexte	53
2.8.5	Choix architecturaux et interactions entre les dimensions	54
2.9	Évaluation des systèmes RAG classiques	54
2.10	Limites des métriques classiques face aux systèmes Agentic RAG	55
2.11	Pipeline d'évaluation structuré des systèmes Agentic RAG	55
2.11.1	Évaluation au niveau des composants	56
2.11.1.0.1	Planificateur :	56
2.11.1.0.2	Moteur de récupération :	56
2.11.1.0.3	Outils externes :	57
2.11.2	Évaluation au niveau des trajectoires	57
2.11.3	Évaluation au niveau système	58
2.12	Limites, modes de défaillance et défis de fiabilité	58
2.12.0.1	Dérive de récupération et désalignement des requêtes	59
2.12.0.2	Mauvaise utilisation des outils et erreurs en cascade	59

2.12.0.3 Injection de prompt	59
--	----

3 Conception et Réalisation de la Solution 60

3.1 Introduction	60
3.2 Analyse du contexte	60
3.3 Étude de l'existant	61
3.3.1 Solutions commerciales de BI conversationnelle	62
3.3.2 Approches académiques de type Text-to-SQL / NL2SQL	62
3.3.3 Travaux de recherche sur l'Agentic RAG	62
3.4 Présentation du système proposé	63
3.5 Conception de l'architecture du système	64
3.5.1 Formalisation du système proposé	64
3.5.2 Modules principaux de l'architecture	69
3.5.2.1 Classification d'intention	69
3.5.2.2 Récupération sémantique des schémas	70
3.5.2.3 Agent Orchestrateur	70
3.5.2.4 Agents de Génération de Requêtes	71
3.5.2.5 Environnement d'Exécution et Validation	72
3.5.2.6 Module de Fusion et Génération de Réponse	72
3.6 Pipeline d'exécution du système	72
3.7 Implémentation de la Solution	76
3.7.1 Méthodologie de développement	76
3.7.1.1 Préparation des données	76
3.7.1.2 Indexation des schémas	76
3.7.1.3 Agent de classification d'intention	77
3.7.1.4 Récupération sémantique des schémas	77
3.7.1.5 Agent orchestrateur	78
3.7.1.6 Agents de génération de requêtes	78
3.7.1.7 Agent de validation	79
3.7.1.8 Agent d'exécution	80

3.7.1.9	Agent de fusion	80
3.7.1.10	Agent de visualisation	81
3.7.1.11	Agent d’insight et de génération de réponse	81
3.7.2	Ingénierie des Prompts	82
3.7.2.1	Assignment de rôle	82
3.7.2.2	Injection du contexte structuré	82
3.7.2.3	Règles métier explicites et contraintes négatives	83
3.7.2.4	Exemples few-shot ciblés	83
3.7.2.5	Contrôle du format de sortie et parsing défensif	83
3.7.2.6	Prompt d’erreur réflexif pour le query rewriting	84
3.7.2.7	Séparation fait/interprétation dans l’agent Insight	84
3.8	Études de cas et résultats préliminaires	85
3.8.1	Étude de Cas 1 : Évolution mensuelle du revenu total en 2022	85
3.8.2	Étude de Cas 2 : Analyse du chiffre d’affaires par wilaya en 2023	87
3.8.3	Conclusion	88
3.9	Évaluation du système Agentic RAG	88
3.9.1	Protocole d’évaluation	88
3.9.2	Batch 1 : Requêtes simples	89
3.9.3	Batch 2 : Requêtes intermédiaires	90
3.9.4	Batch 3 : Requêtes complexes	91
3.9.5	Synthèse globale	92
3.9.6	Conclusion	93
3.10	Limites et discussion	93
3.10.1	Limites identifiées	93
3.10.2	Perspectives et travaux futurs	94
	Conclusion générale	95
	Bibliographie	97
	Appendices	101

.1	Ingénierie des prompts (Prompt Engineering)	101
.1.1	Définition et importance	101
.1.2	Taxonomie des techniques de prompting	101
.1.3	Optimisation automatique des prompts	102
.1.4	Bonnes pratiques et principes de conception	102
.2	Architecture et Choix Technologiques	103
.2.1	Large Language Model (LLM)	103
.2.2	Framework d'Orchestration Multi-Agents	104
.2.3	Modèle d'Embedding et Index Vectoriel	105

Liste des tableaux

1.1	Comparatif des modèles NoSQL et de leurs représentants (Source : adapté de [1])	25
2.1	Comparaison des flux de raisonnement (<i>Reasoning Workflows</i>)	48
2.2	Composants du tuple des systèmes agentiques basés sur les LLM	48
3.1	Classification du système selon les dimensions taxonomiques	67
3.2	Évolution mensuelle du revenu en 2022	86
3.3	Chiffre d'affaires par wilaya en 2023	87
3.4	Performances globales — Batch 1 (toutes couches)	90
3.5	Performances globales — Batch 2 (toutes couches)	91
3.6	Performances globales — Batch 3 (toutes couches)	92
3.7	Tableau comparatif et synthèse globale des performances du système	93
8	Espace des alternatives pour le choix du LLM	103
9	Espace des alternatives pour les frameworks d'orchestration	105

Table des figures

1.1	Modèle DIKW représentant la hiérarchie entre données, information, connaissance et sagesse	21
1.2	Schéma d'une base de données relationnelle	23
1.3	Architecture et typologie des bases de données NoSQL	23
1.4	Schéma d'une base de données orientée documents	24
1.5	Schéma d'une base de données orientée graphe	25
1.6	Architecture générale d'une solution BI : du système opérationnel aux outils d'analyse, d'après [2])	27
1.7	Architecture générale du Transformer : encodeur (gauche) et décodeur (droite), d'après [3]	34
1.8	Schéma du mécanisme Scaled Dot-Product Attention	34
1.9	Schéma de l'attention multi-têtes	35
2.1	Architecture générale d'un système RAG, d'après [4]	38
2.2	Évolution des paradigmes RAG : du Naive RAG au Modular RAG, d'après [4] .	40
2.3	Schéma des approches basées sur le routage, d'après [5]	44
2.4	Schéma des approches itératives en boucle, d'après [5]	45
2.5	Schéma des approches hybrides modulaires, d'après [4]	46
2.6	Taxonomie de l'Agentic RAG, d'après[6].	51
2.7	Taxonomie du raisonnement	53
3.1	Présentation du système proposé	63
3.2	Vue d'ensemble de l'architecture	64
3.3	Schéma du processus de récupération itérative	68
3.4	Paradigme de raisonnement : ReAct	69
3.5	Pipeline d'exécution du système	75

3.6	Graphique de l'évolution mensuelle du revenu en 2022	86
3.7	Graphique du chiffre d'affaires par wilaya en 2023	88

Liste des acronymes

- **ACID** : Atomicity, Consistency, Isolation, Durability
- **API** : Application Programming Interface
- **BI** : Business Intelligence
- **BSON** : Binary JSON
- **CoT** : Chain-of-Thought
- **CRM** : Customer Relationship Management
- **CSV** : Comma-Separated Values
- **DIKW** : Data, Information, Knowledge, Wisdom
- **ELT** : Extract, Load, Transform
- **ERP** : Enterprise Resource Planning
- **ETL** : Extract, Transform, Load
- **FAISS** : Facebook AI Similarity Search
- **HOLAP** : Hybrid Online Analytical Processing
- **IA** : Intelligence Artificielle
- **IoT** : Internet of Things
- **JSON** : JavaScript Object Notation
- **KPI** : Key Performance Indicator
- **LLM** : Large Language Model
- **MOLAP** : Multidimensional Online Analytical Processing
- **NoSQL** : Not Only SQL
- **OLAP** : Online Analytical Processing
- **RAG** : Retrieval-Augmented Generation
- **RDBMS** : Relational Database Management System
- **ReAct** : Reasoning and Acting
- **RDF** : Resource Description Framework
- **ROLAP** : Relational Online Analytical Processing
- **SQL** : Structured Query Language
- **XML** : eXtensible Markup Language

Introduction générale

Contexte et Enjeux

Avec la transformation numérique des entreprises et des administrations, les volumes de données générés quotidiennement connaissent une croissance considérable. Ces données proviennent de sources variées et hétérogènes : bases de données relationnelles, fichiers CSV, systèmes NoSQL, entrepôts de données analytiques ou encore plateformes métier. Dans le secteur des télécommunications en particulier, les opérateurs génèrent quotidiennement des millions de transactions couvrant les recharges, les ventes, les souscriptions et les revenus clients, réparties sur l'ensemble du territoire national.

Malgré cette abondance d'informations, l'exploitation efficace de ces données demeure une tâche complexe, réservée aux profils techniques maîtrisant les langages de requêtes, la modélisation de données et les outils d'analyse spécialisés. Les solutions analytiques traditionnelles, bien qu'elles permettent la création de tableaux de bord et d'indicateurs de performance, reposent sur des structures prédéfinies et nécessitent une forte intervention humaine pour la formulation des requêtes et l'interprétation des résultats.

En parallèle, les avancées récentes en intelligence artificielle, notamment les **Large Language Models (LLMs)** comme GPT-4, et les systèmes **Retrieval-Augmented Generation (RAG)**, ouvrent de nouvelles perspectives pour l'interaction avec les données en langage naturel. Ces technologies permettent d'automatiser certaines tâches analytiques et de rendre les systèmes d'aide à la décision accessibles à des utilisateurs non techniques. C'est dans ce contexte technologique en pleine évolution que s'inscrit le présent travail.

Problématique

Si les LLMs ont considérablement amélioré les capacités de compréhension du langage naturel, ils présentent une limite fondamentale : ils reposent sur des données statiques acquises lors de leur phase d'entraînement et ne disposent pas d'un accès direct aux données métier actualisées, privées ou spécifiques à une entreprise. Cette contrainte réduit leur fiabilité dans les contextes où la précision, l'actualité et la contextualisation des informations sont indispensables.

Les architectures RAG ont émergé pour pallier cette limitation en combinant génération de texte et récupération d'informations externes. Cependant, les approches RAG traditionnelles reposent sur des architectures monolithiques à agent unique, chargé à la fois de comprendre la requête, d'interroger les sources et de formuler la réponse. Ces architectures atteignent rapidement leurs

limites face à des environnements de données hétérogènes, multi-sources et distribués. Elles nécessitent par ailleurs des *prompts* surchargés en schémas et en exemples, ce qui entraîne une consommation accrue de ressources, des temps de traitement élevés et une précision parfois dégradée.

Face à ces constats, les architectures multi-agents et l' **Agentic RAG** apparaissent comme une réponse prometteuse. En répartissant les responsabilités entre plusieurs agents spécialisés, ces systèmes offrent une modularité accrue, une meilleure scalabilité et une capacité supérieure à traiter des environnements de données complexes et distribués.

Problématique centrale : *Comment concevoir un système intelligent capable d'interroger automatiquement des sources de données hétérogènes en langage naturel, puis de produire de manière autonome des analyses pertinentes, des visualisations et des réponses contextualisées ?*

Objectifs

L'objectif principal de ce mémoire est de concevoir et d'implémenter un chatbot intelligent capable de générer automatiquement des visualisations et des analyses business afin d'aider à la prise de décision.

Plus spécifiquement, la solution vise à :

- Permettre aux utilisateurs d'interagir avec les données en langage naturel, sans compétences techniques particulières .
- Transformer les requêtes utilisateurs en analyses exploitables à partir de données business .
- Générer automatiquement des visualisations pertinentes (graphes, courbes, indicateurs) selon les données interrogées .
- Produire des insights clairs et compréhensibles pour faciliter la prise de décision ;
- Proposer une solution simple, interactive et orientée aide à la décision pour les données d'entreprise.

Ce mémoire est structuré en 3 chapitres de la manière suivante :

Chapitre 1 : Ce chapitre pose les fondements théoriques et l'état de l'art nécessaires à la compréhension de notre travail. Il couvre les concepts de la Business Intelligence, l'Intelligence Artificielle , ainsi que les Grands Modèles de Langage (LLM) et leur rôle central dans les systèmes modernes de traitement du langage naturel.

Chapitre 2 : Ce chapitre présente une étude conceptuelle approfondie du RAG Agentique. Il retrace l'évolution de la Génération Augmentée par Récupération (RAG), détaille ses

modules constitutifs, puis établit une distinction fondamentale entre le raisonnement pré-défini et le raisonnement agentique. Il aboutit à une formalisation rigoureuse du RAG Agentique, en précisant ses composantes, ses mécanismes d'orchestration et les approches d'évaluation adaptées à ce paradigme.

Chapitre 3 : Ce chapitre est consacré à la conception et au développement de notre solution. Il décrit l'architecture globale du système, une description pour chaque module ainsi que le protocole d'évaluation en trois niveaux adopté. Il se conclut par une analyse et une discussion des résultats obtenus.

Enfin, nous terminerons ce mémoire par les principales conclusions et quelques perspectives.

Chapitre 1

État de l'art

Introduction

Ce chapitre présente les fondements théoriques et technologiques sur lesquels repose ce travail de mémoire. Il a pour objectif d'introduire les concepts essentiels liés à la gestion intelligente des données ainsi qu'aux approches récentes de l'intelligence artificielle utilisées dans les systèmes conversationnels modernes.

Dans un premier temps, nous nous intéressons à la Business Intelligence (BI), domaine consacré à la collecte, l'intégration, l'analyse et la visualisation des données afin de soutenir la prise de décision au sein des organisations. Cette section aborde les architectures décisionnelles, les entrepôts de données, les processus ETL ainsi que les principales technologies utilisées pour transformer des données brutes en informations exploitables.

Dans un second temps, nous présentons les notions fondamentales relatives à l'Intelligence Artificielle, en mettant particulièrement l'accent sur les modèles récents de traitement du langage naturel. Nous abordons notamment l'architecture des Transformers, les Large Language Models (LLMs), ainsi que les techniques d'ingénierie des prompts permettant d'optimiser les interactions avec ces modèles. Ces concepts constituent aujourd'hui le socle des systèmes conversationnels intelligents capables d'interpréter et de générer du langage naturel avec un haut niveau de performance.

L'ensemble de ces éléments théoriques permettra ainsi de mieux comprendre les choix technologiques et méthodologiques adoptés dans la conception de notre système d'analyse de données conversationnel polyglotte.

De l'information à la décision

1.1 Donnée, information et décision

1.1.1 Donnée

La donnée représente une description élémentaire d'un objet, d'un événement ou d'une transaction. Elle correspond à une codification structurée de faits observables. Dans le contexte d'une entreprise commerciale, les données peuvent, par exemple, concerner les clients, les produits, les ventes ou les points de distribution.

Les données sont généralement brutes, isolées et dépourvues de signification immédiate lorsqu'elles sont considérées indépendamment de leur contexte. Elles constituent ainsi la base sur laquelle reposent les traitements analytiques et les systèmes décisionnels [7].

1.1.2 Information

L'information est le résultat du traitement, de l'organisation et de l'interprétation des données. Contrairement à la donnée brute, l'information possède une signification dans un contexte précis et peut être mobilisée par les décideurs pour comprendre une situation ou évaluer une activité.

À titre d'illustration, le nombre total de ventes réalisées au cours d'un mois constitue une donnée. En revanche, l'évolution de ces ventes par rapport au mois précédent constitue une information, dans la mesure où elle permet d'interpréter les performances de l'entreprise. L'information apporte ainsi une valeur ajoutée aux données en facilitant l'analyse et la compréhension des phénomènes étudiés [7].

1.1.3 Connaissance et décision

La connaissance émerge lorsque les informations sont exploitées par les décideurs afin d'orienter leurs actions. Elle résulte non seulement de l'analyse des informations disponibles, mais également de l'expérience et de l'expertise des responsables.

Dans un environnement décisionnel, l'objectif principal est donc de transformer les données en informations pertinentes, puis ces informations en connaissances exploitables pour soutenir la prise de décision aux niveaux stratégique, tactique et opérationnel. Par exemple, l'analyse des habitudes d'achat peut révéler une diminution des ventes dans une région donnée à la suite de l'ouverture d'un concurrent. Cette connaissance peut conduire l'entreprise à déployer de nouvelles stratégies commerciales afin de fidéliser sa clientèle [7].

1.2 Types de données

Avec l'évolution des systèmes d'information et l'essor du Big Data, les données manipulées par les organisations prennent différentes formes et structures. On distingue généralement trois

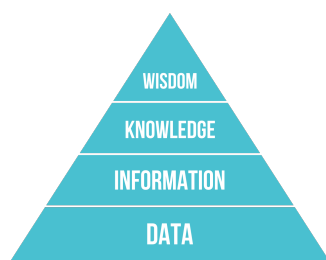


FIG. 1.1 : Modèle DIKW représentant la hiérarchie entre données, information, connaissance et sagesse

catégories : les données structurées, semi-structurées et non structurées.

1.2.1 Données structurées

Les données structurées sont organisées selon un modèle précis et prédéfini. Elles respectent une structure tabulaire composée de lignes et de colonnes, ce qui facilite leur stockage, leur recherche et leur traitement.

Ce type de données est principalement utilisé dans les bases de données relationnelles et les tableurs. Chaque champ possède un type bien défini (texte, nombre, date, etc.), permettant une manipulation efficace à l'aide de langages de requête tels que SQL. Les données structurées sont particulièrement adaptées aux systèmes transactionnels et aux outils de Business Intelligence traditionnels .

1.2.2 Données semi-structurées

Les données semi-structurées possèdent une organisation partielle sans respecter un schéma rigide comparable à celui des bases relationnelles. Elles contiennent généralement des balises, des métadonnées ou des marqueurs permettant de décrire leur structure, offrant ainsi davantage de flexibilité tout en conservant une certaine organisation.

Parmi les formats courants figurent les fichiers XML, JSON et les documents RDF. Les données semi-structurées sont largement utilisées dans les échanges inter-applicatifs, les services web et les systèmes distribués.

1.2.3 Données non structurées

Les données non structurées ne suivent aucun modèle prédéfini et se présentent sous forme brute. Elles représentent aujourd'hui la majorité des données générées sur Internet et dans les systèmes modernes.

Ces données sont souvent difficiles à stocker, à analyser et à exploiter à l'aide des outils classiques de gestion de bases de données. On y retrouve notamment les documents texte, les images, les vidéos, les fichiers audio, les publications sur les réseaux sociaux, les pages web et les courriers électroniques. L'essor de l'intelligence artificielle et du traitement automatique du

langage naturel a considérablement renforcé les capacités d'analyse de ce type de données [8].

1.3 Types de bases de données

Les bases de données constituent un élément fondamental des systèmes d'information modernes. Elles permettent de stocker, d'organiser, de gérer et d'exploiter les données de manière efficace afin de répondre aux besoins opérationnels et décisionnels des organisations. Avec l'évolution des technologies numériques, l'augmentation massive des volumes de données et la diversification des formats de stockage, plusieurs modèles ont été développés pour répondre à des problématiques spécifiques liées à la performance, la flexibilité, la scalabilité et l'analyse des données.

Traditionnellement, les systèmes d'information reposaient principalement sur les bases de données relationnelles. Cependant, l'émergence du Big Data, des applications web à grande échelle, de l'Internet des objets (IoT) et des réseaux sociaux a conduit à l'apparition de nouveaux modèles dits NoSQL, capables de gérer des données volumineuses, hétérogènes et distribuées [1].

1.3.1 Bases de données relationnelles

Les bases de données relationnelles, désignées par l'acronyme RDBMS (Relational Database Management Systems), représentent le modèle le plus répandu dans les systèmes d'information traditionnels. Introduit dans les années 1970, ce modèle repose sur une organisation tabulaire des données sous forme de relations composées de lignes et de colonnes [8].

Dans une base relationnelle, chaque table représente une entité spécifique du système d'information. Chaque ligne correspond à un enregistrement unique, tandis que chaque colonne représente un attribut de cette entité. Le modèle s'appuie sur plusieurs concepts fondamentaux : la clé primaire permettant d'identifier de manière unique chaque enregistrement, les clés étrangères établissant des relations entre tables, les contraintes d'intégrité garantissant la cohérence des données, et le langage SQL permettant leur manipulation et leur interrogation.

Les bases relationnelles privilégient une organisation normalisée des données. La normalisation consiste à répartir les informations dans plusieurs tables afin d'éviter la redondance et les incohérences lors des opérations de mise à jour.

L'un des principaux avantages des systèmes relationnels réside dans leur conformité aux propriétés ACID :

- **Atomicité** : une transaction est exécutée dans sa totalité ou annulée ;
- **Cohérence** : les données demeurent valides après chaque transaction ;
- **Isolation** : les transactions concurrentes n'interfèrent pas entre elles ;
- **Durabilité** : les modifications validées sont conservées de manière permanente.

Ces propriétés rendent les bases relationnelles particulièrement adaptées aux applications transactionnelles critiques telles que les systèmes bancaires, les plateformes de commerce électronique ou les systèmes de gestion des ressources humaines [8].

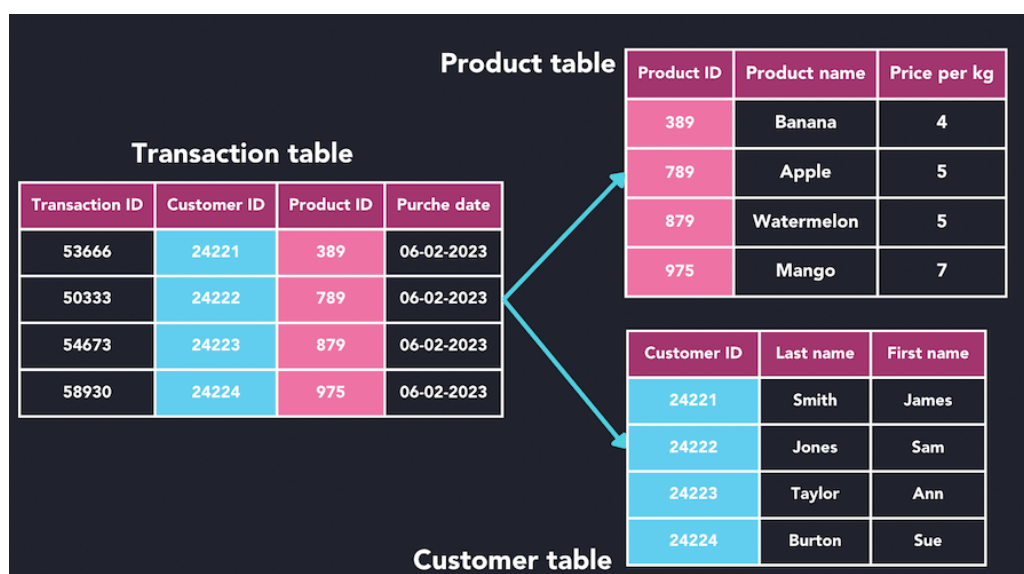


FIG. 1.2 : Schéma d'une base de données relationnelle

Malgré leur robustesse, les bases relationnelles présentent certaines limites face aux environnements Big Data. Leur structure rigide rend difficile la gestion de données hétérogènes ou en évolution rapide, et la montée en charge horizontale peut s'avérer complexe dans les systèmes distribués à très grande échelle.

1.3.2 Bases de données non relationnelles (NoSQL)

Les bases de données NoSQL sont apparues au début des années 2000 pour répondre aux limitations des bases relationnelles face à l'augmentation massive du volume, de la variété et de la vitesse des données générées par les applications web modernes. Le terme NoSQL désigne un ensemble de systèmes non relationnels conçus pour les environnements distribués et les architectures à grande échelle [1].

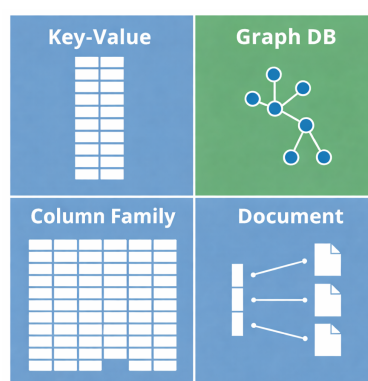
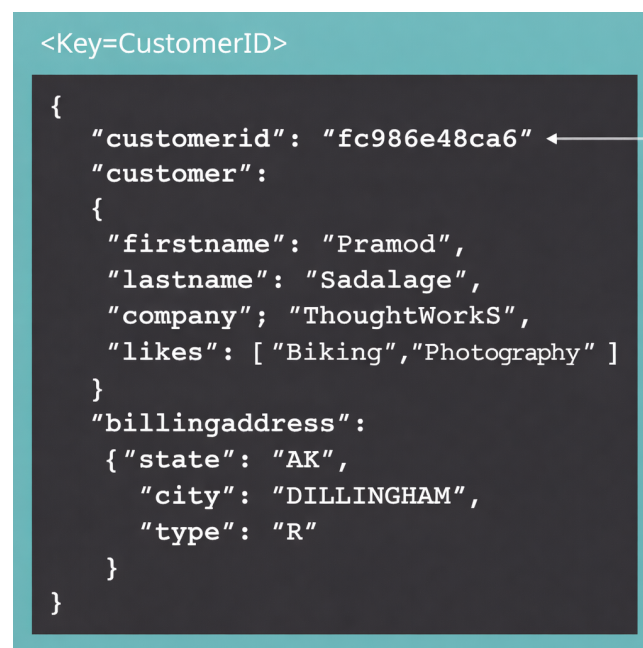


FIG. 1.3 : Architecture et typologie des bases de données NoSQL

Contrairement aux bases relationnelles, les systèmes NoSQL n'imposent généralement pas de schéma rigide et privilégient la flexibilité, la scalabilité horizontale et les performances sur de grands volumes de données. Ils abandonnent parfois certaines propriétés des systèmes relationnels, notamment la cohérence forte et les transactions ACID, afin d'améliorer la disponibilité et la rapidité d'accès aux données.

1.3.2.0.1 Bases clé-valeur. Les bases clé-valeur représentent le modèle NoSQL le plus simple. Les données y sont stockées sous forme de paires composées d'une clé unique et d'une valeur associée, dont le contenu peut être du texte, du JSON, du XML ou des données binaires. Ce modèle offre des performances élevées grâce à un accès direct par clé et une mise à l'échelle facilitée par le partitionnement (sharding). Il est principalement utilisé pour le stockage de sessions utilisateur, les systèmes de cache ou les préférences utilisateurs. En contrepartie, ces bases restent limitées pour les requêtes complexes, les jointures ou les transactions multi-opérations. [1].

1.3.2.0.2 Bases orientées documents. Les bases orientées documents stockent les données sous forme de documents JSON, BSON ou XML. Contrairement aux bases relationnelles, les documents d'une même collection peuvent présenter des structures différentes, offrant une grande flexibilité pour les données semi-structurées et évolutives. Ces bases sont particulièrement adaptées aux systèmes de gestion de contenu, aux applications web modernes et à l'analyse de données en temps réel. [1].



```
<Key=CustomerID>
{
  "customerid": "fc986e48ca6"
  "customer":
  {
    "firstname": "Pramod",
    "lastname": "Sadhalage",
    "company": "ThoughtWorkS",
    "likes": [ "Biking", "Photography" ]
  }
  "billingaddress":
  {
    "state": "AK",
    "city": "DILLINGHAM",
    "type": "R"
  }
}
```

FIG. 1.4 : Schéma d'une base de données orientée documents

1.3.2.0.3 Bases orientées colonnes. Les bases orientées colonnes organisent les données sous forme de familles de colonnes plutôt que de lignes traditionnelles. Elles sont conçues pour gérer de très grands volumes de données distribuées avec une forte disponibilité et de hautes performances en écriture. Ces bases sont particulièrement utilisées pour les traitements massifs de données, les plateformes Big Data ou les applications IoT [1].

1.3.2.0.4 Bases de données orientées graphes. Les bases de données graphes sont conçues pour représenter et analyser les relations entre entités. Elles modélisent les informations sous forme de nœuds (entités) et d'arêtes (relations), directement stockées dans la structure du graphe, permettant des parcours relationnels très rapides. Elles sont particulièrement adaptées aux réseaux sociaux, aux moteurs de recommandation et à la détection de fraude [1].

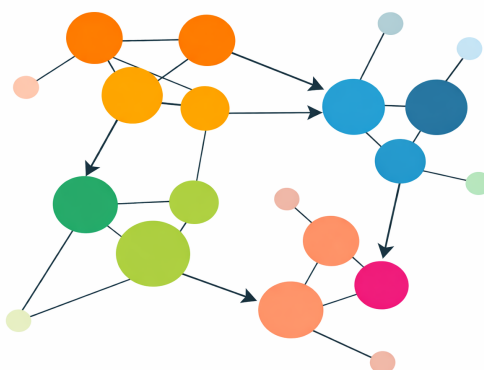


FIG. 1.5 : Schéma d'une base de données orientée graphe

Le tableau suivant synthétise les principaux modèles NoSQL et leurs représentants :

TAB. 1.1 : Comparatif des modèles NoSQL et de leurs représentants (Source : adapté de [1])

Modèle de Données	Exemples de Bases de Données
Key-Value (Clé-Valeur)	BerkeleyDB, LevelDB, Memcached, Project Voldemort, Redis, Riak
Document	CouchDB, MongoDB, OrientDB, RavenDB, Terrastore
Column-Family (Colonnes)	Amazon SimpleDB, Cassandra, HBase, Hypertable
Graph (Graphe)	FlockDB, HyperGraphDB, Infinite Graph, Neo4J, OrientDB

Aujourd'hui, bases relationnelles et NoSQL sont fréquemment utilisées de manière complémentaire dans une approche dite de persistance polyglotte, où plusieurs technologies de stockage coexistent au sein d'un même système d'information selon les contraintes métier et techniques [1].

1.4 Contexte et définition de la Business Intelligence

1.4.1 Contexte

L'évolution des systèmes d'information et des technologies numériques a profondément transformé le fonctionnement des organisations modernes. La majorité des entreprises disposent

désormais de systèmes informatiques permettant de gérer de manière automatisée différentes fonctions telles que la gestion commerciale, les ressources humaines, la finance ou la logistique.

Sur le plan économique, la mondialisation et l'intensification de la concurrence imposent aux organisations une adaptation permanente. Elles doivent être capables d'analyser rapidement leur environnement, d'identifier les évolutions du marché et de prendre des décisions stratégiques efficaces dans un contexte en constante mutation. Parallèlement, les avancées technologiques ont engendré une croissance considérable des volumes de données générées quotidiennement, en provenance de sources multiples et hétérogènes. Cette multiplication des données rend leur exploitation de plus en plus complexe [7].

1.4.2 Définition de la Business Intelligence

La Business Intelligence (BI), également désignée par le terme informatique décisionnelle, regroupe l'ensemble des moyens, méthodes et outils permettant de soutenir les processus de collecte, de consolidation, de modélisation, d'analyse et de restitution des informations au sein d'une organisation.

Elle constitue une interface entre les systèmes de stockage de données et les besoins décisionnels des entreprises. L'objectif principal de la BI est de transformer les données brutes en informations exploitables afin d'aider les décideurs à piloter efficacement les activités de l'entreprise [7].

1.4.3 Objectifs de la Business Intelligence

L'informatique décisionnelle vise principalement à améliorer la qualité et la rapidité de la prise de décision au sein des organisations. Ses principaux objectifs sont les suivants [7] :

- faciliter la prise de décision stratégique, tactique et opérationnelle ;
- transformer les données en informations exploitables et compréhensibles ;
- analyser les performances et identifier les tendances du marché ;
- améliorer le pilotage des activités de l'entreprise ;
- fournir des tableaux de bord et des outils de visualisation adaptés aux décideurs ;
- exploiter des données provenant de sources multiples et hétérogènes ;
- optimiser la réactivité et la compétitivité de l'organisation.

1.5 Architecture et fonctionnement d'une solution BI

1.5.1 Introduction

Une solution de Business Intelligence permet de collecter, d'intégrer, de stocker, d'analyser et de restituer les données provenant de différentes sources sous une forme compréhensible et utile à la prise de décision.

Une architecture BI repose sur plusieurs composants complémentaires fonctionnant de manière coordonnée : les sources de données, le processus ETL, l'entrepôt de données, et les outils d'analyse et de visualisation. L'ensemble de ces composants produit des indicateurs et des analyses permettant une meilleure compréhension de l'activité de l'entreprise.

1.5.2 Architecture générale

De manière générale, une solution BI s'articule autour des composants suivants : les sources de données, le processus ETL (ou ELT), l'entrepôt de données, et les outils d'analyse et de restitution.

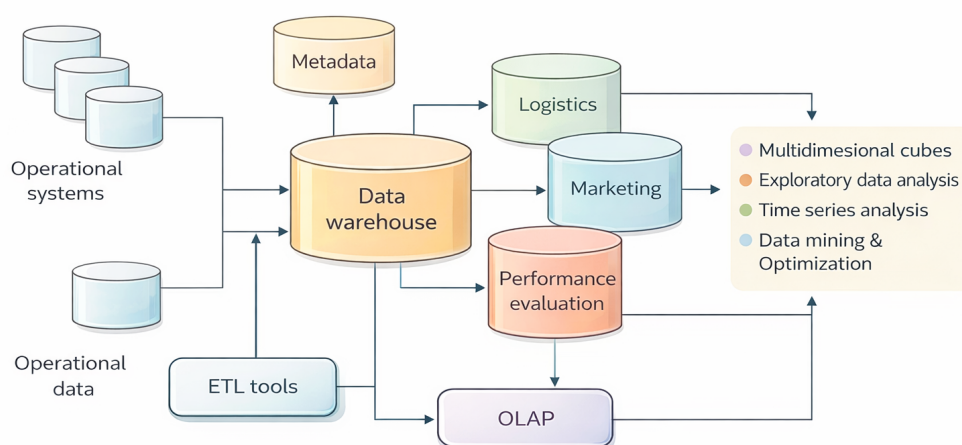


FIG. 1.6 : Architecture générale d'une solution BI : du système opérationnel aux outils d'analyse, d'après [2])

Les données issues des différentes sources sont d'abord extraites et déposées dans une zone tampon (*staging*), puis transformées et chargées dans l'entrepôt de données. Une fois centralisées et organisées, elles peuvent être exploitées à travers différents outils d'analyse, de reporting et de visualisation afin de produire des tableaux de bord et des indicateurs d'aide à la décision.

1.5.3 Les sources de données

Les données exploitées dans une solution BI peuvent provenir de sources internes ou externes à l'entreprise, notamment : les bases de données relationnelles (Oracle, MySQL, SQL Server), les fichiers plats (Excel, CSV), les fichiers semi-structurés (XML, JSON), les applications métiers (ERP, CRM, systèmes de gestion des ressources humaines), ainsi que les données issues du web, des réseaux sociaux ou des capteurs connectés (IoT).

Ces données sont souvent hétérogènes, tant au niveau de leur structure que de leur format. Cette diversité rend leur exploitation directe difficile et nécessite une phase de préparation et d'intégration avant leur utilisation dans le système décisionnel [2].

1.5.4 L'entrepôt de données (Data Warehouse)

L'entrepôt de données, ou Data Warehouse, constitue le composant central d'une architecture BI. Bill Inmon, considéré comme le père du concept, le définit comme une collection de données orientée sujet, intégrée, non volatile et historisée, conçue pour supporter les processus de prise de décision [2].

Ces quatre caractéristiques signifient concrètement que :

- les données sont organisées autour des grands sujets métiers de l'entreprise (clients, ventes, produits), plutôt qu'autour des applications qui les produisent ;
- elles sont intégrées et homogénéisées à partir de sources hétérogènes ;
- elles ne sont pas modifiées une fois chargées (propriété de non-volatilité) ;
- elles conservent un historique permettant l'analyse dans le temps.

1.5.4.1 Deux approches de conception

Deux méthodologies s'affrontent dans la conception d'un entrepôt de données.

L'approche *top-down* d'Inmon préconise la construction d'un entrepôt centralisé et normalisé en premier lieu, à partir duquel sont ensuite dérivés des datamarts départementaux. Cette méthode est plus longue à mettre en œuvre, mais offre une meilleure cohérence globale des données [2].

L'approche *bottom-up* de Kimball préconise au contraire de construire les datamarts en premier, chacun modélisé selon un schéma dimensionnel, puis de les intégrer progressivement. Elle est réputée pour sa rapidité de déploiement et sa lisibilité pour les utilisateurs métiers [9].

1.5.4.2 Modèles de conception dimensionnelle

Dans l'approche Kimball, les données sont organisées selon des modèles dimensionnels structurés autour d'une table de faits centrale et de tables de dimensions contenant les attributs descriptifs (temps, produits, régions). Deux variantes existent [9] :

1.5.4.2.1 Le schéma en étoile (star schema) : les tables de dimensions sont directement reliées à la table de faits. Les requêtes sont simples et rapides, car elles nécessitent peu de jointures.

1.5.4.2.2 Le schéma en flocon (snowflake schema) : les tables de dimensions sont elles-mêmes normalisées en sous-tables. Cette structure réduit la redondance et améliore l'efficacité du stockage, au prix de requêtes plus complexes.

1.5.4.3 Datamarts

L'entrepôt est généralement subdivisé en datamarts, des sous-ensembles thématiques orientés vers un domaine d'activité précis (ventes, finances, ressources humaines). Ils permettent d'offrir aux équipes métiers une vue filtrée et agrégée des données, optimisée pour leurs besoins spécifiques d'analyse [9].

1.5.5 Le processus ETL

Le processus ETL (Extract – Transform – Load) joue un rôle essentiel dans une solution BI. Il assure l'intégration des données provenant des différentes sources vers l'entrepôt de données en trois étapes successives.

1.5.5.1 Extraction (Extract)

L'étape d'extraction consiste à récupérer les données depuis les différentes sources identifiées. Cette opération peut être réalisée de deux manières : une extraction complète, où l'ensemble des données de la source est récupéré, ou une extraction incrémentale, où seules les nouvelles données ou les données modifiées depuis la dernière extraction sont récupérées. Les données extraites sont temporairement stockées dans une zone de *staging*, espace tampon intermédiaire permettant de les isoler et de les préparer avant transformation, sans impacter les systèmes sources.

1.5.5.2 Transformation (Transform)

Après l'extraction, les données subissent plusieurs opérations de transformation afin de garantir leur cohérence, leur qualité et leur compatibilité avec l'entrepôt. Ces opérations consistent notamment à corriger les formats de données, supprimer les doublons, convertir les dates et les unités, fusionner ou découper certains champs, et calculer des valeurs dérivées. Cette étape est déterminante, car elle conditionne la fiabilité des données pour les analyses décisionnelles.

1.5.5.3 Chargement (Load)

Une fois transformées, les données sont chargées dans l'entrepôt. Le chargement peut être initial (premier remplissage), incrémental (ajout périodique de nouvelles données) ou complet

(remplacement total des données existantes).

1.5.5.4 ETL vs ELT

Il convient de distinguer l'ETL de l'ELT (Extract – Load – Transform), un paradigme apparu avec les entrepôts de données cloud (Snowflake, Google BigQuery, Amazon Redshift). Dans l'ELT, les données sont d'abord chargées brutes dans l'entrepôt, puis transformées directement en son sein en exploitant sa puissance de calcul. Cette approche est aujourd'hui largement adoptée dans les architectures modernes, car elle offre davantage de flexibilité et de scalabilité [8].

1.5.6 Les outils d'analyse et de visualisation

Une fois les données centralisées dans l'entrepôt, différents outils permettent leur exploitation afin de produire des analyses, des indicateurs et des visualisations utiles à la prise de décision.

1.5.6.1 Reporting

Les outils de reporting permettent de générer des rapports et des représentations graphiques facilitant l'interprétation des données. Ils offrent une vue synthétique des performances de l'entreprise et permettent de suivre l'évolution des indicateurs dans le temps.

1.5.6.2 OLAP

Les outils OLAP (Online Analytical Processing) permettent l'analyse multidimensionnelle des données. Les informations sont organisées sous forme de cubes multidimensionnels afin de les comparer selon plusieurs dimensions, telles que le temps, les produits ou les régions. On distingue trois variantes principales [7] :

1.5.6.2.1 ROLAP (Relational OLAP) : les cubes sont simulés à partir de bases de données relationnelles. Cette approche offre une bonne scalabilité, mais des performances moindres sur de très grands volumes.

1.5.6.2.2 MOLAP (Multidimensional OLAP) : les données sont précalculées et stockées dans des structures multidimensionnelles natives, offrant de très hautes performances de requête.

1.5.6.2.3 HOLAP (Hybrid OLAP) : combinaison des deux approches précédentes, cherchant à concilier performance et scalabilité.

1.5.6.3 Tableaux de bord et visualisation

Les tableaux de bord regroupent les principaux indicateurs de performance (KPI) sous forme de graphiques et de visualisations interactives. Ils permettent aux décideurs d'obtenir une vision globale et synthétique de l'activité de l'entreprise. Parmi les outils de visualisation les plus utilisés figurent Microsoft Power BI, Tableau et Qlik.

1.5.6.4 Analyse avancée

Certaines solutions BI intègrent des fonctionnalités d'analyse avancée, situées à la frontière entre la BI traditionnelle et la science des données [7], notamment :

- **le data mining** : extraction de *patterns* et de connaissances cachées dans les données ;
- **l'analyse prédictive** : anticipation de comportements ou d'évolutions futures à partir de modèles statistiques et d'apprentissage automatique ;
- **l'analyse statistique** : exploitation de méthodes quantitatives pour interpréter les données ;
- **les technologies Big Data** : traitement de volumes massifs de données via des frameworks distribués tels qu'Apache Hadoop ou Apache Spark.

Ces approches permettent d'extraire des connaissances plus complexes et d'anticiper certaines évolutions, dépassant ainsi le simple reporting historique caractéristique de la BI traditionnelle.

Intelligence artificielle et grands modèles de langage

1.6 Définition et périmètre

L'intelligence artificielle (IA) est un domaine scientifique visant à doter les machines de capacités cognitives telles que l'apprentissage, le raisonnement, la perception et la prise de décision. Elle recouvre un spectre d'approches allant des méthodes symboliques fondées sur des règles explicites aux méthodes statistiques modernes reposant sur les données. Formellement introduit lors de la conférence de Dartmouth en 1956 ce champ a depuis considérablement évolué, des premiers systèmes experts aux agents conversationnels contemporains.

1.7 Apprentissage automatique

1.7.1 Définition

L'apprentissage automatique (*Machine Learning*) est une branche de l'IA dans laquelle un système améliore ses performances sur une tâche donnée à partir de l'expérience, sans être explicitement programmé. Sa définition formelle repose sur trois éléments interdépendants [10] : la tâche

(T), qui désigne le problème à résoudre (classification, régression, détection d'anomalies...); la mesure de performance (P), qui permet d'évaluer quantitativement la qualité des prédictions via des métriques telles que l'exactitude (*accuracy*), le taux d'erreur ou la log-vraisemblance; et l'expérience (E), c'est-à-dire les données à partir desquelles le système apprend, généralement représentées sous forme de vecteurs de caractéristiques $x \in \mathbb{R}^n$. L'évaluation est systématiquement réalisée sur un ensemble de test distinct des données d'entraînement, afin de mesurer la capacité de généralisation du modèle.

1.7.2 Types d'apprentissage automatique

On distingue trois grands paradigmes d'apprentissage.

Dans l'apprentissage supervisé, chaque exemple d'entraînement est associé à un label cible. Le modèle apprend à estimer $p(y|x)$, c'est-à-dire la sortie attendue à partir des données d'entrée, couvrant des tâches telles que la classification d'images ou la prédiction de valeurs numériques.

Dans l'apprentissage non supervisé, aucun label n'est fourni. Le modèle cherche à découvrir la structure latente des données en estimant $p(x)$, notamment par des techniques de clustering ou de réduction de dimensionnalité.

L'apprentissage par renforcement constitue un troisième paradigme dans lequel un agent apprend en interagissant avec un environnement et en maximisant un signal de récompense cumulatif, avec des applications en robotique et dans les systèmes de décision séquentielle. Il existe également des approches intermédiaires, telles que l'apprentissage semi-supervisé, qui combine données étiquetées et non étiquetées afin de tirer parti de larges corpus partiellement annotés.

1.8 Apprentissage profond

1.8.1 Concepts fondamentaux

L'apprentissage profond (*Deep Learning*) est une sous-discipline de l'apprentissage automatique reposant sur des réseaux de neurones à plusieurs couches. Chaque couche transforme la représentation produite par la précédente, ce qui permet au réseau d'apprendre progressivement des caractéristiques de plus en plus abstraites [11].

Le composant de base de ces réseaux est le neurone artificiel : il calcule une somme pondérée de ses entrées, puis applique une fonction d'activation non linéaire. C'est cette non-linéarité, combinée à la profondeur du réseau, qui permet de modéliser des relations complexes.

L'entraînement repose sur la rétropropagation du gradient (*backpropagation*), qui ajuste les poids du réseau afin de minimiser une fonction de perte. Deux difficultés courantes accompagnent cet entraînement : le surapprentissage (*overfitting*), limité par des techniques comme le *dropout*, et le gradient évanescent, atténué par des fonctions d'activation modernes comme ReLU [11].

L'essor du Deep Learning depuis les années 2010 s'explique principalement par trois facteurs : la disponibilité de grands jeux de données étiquetées, l'augmentation de la puissance de calcul

(GPU), et le développement de nouvelles architectures.

1.8.2 Architectures de réseaux de neurones profonds

Plusieurs architectures se sont développées pour répondre à des besoins spécifiques :

- **Les réseaux de neurones convolutifs (CNN)** : Adaptés au traitement d'images, ils utilisent des filtres convolutifs pour détecter des caractéristiques visuelles à différentes échelles [12].
- **Les réseaux de neurones récurrents (RNN)** : Conçus pour les données séquentielles, ils maintiennent un état caché au fil du temps. Leurs variantes LSTM et GRU ont longtemps dominé le traitement du langage naturel, avant d'être remplacées par les Transformers [13].
- **Les autoencodeurs** : Composés d'un encodeur et d'un décodeur, ils compressent puis reconstruisent les données. Ils sont utilisés en réduction de dimensionnalité et en détection d'anomalies.
- **Les réseaux antagonistes génératifs (GAN)** : Ils mettent en compétition un générateur et un discriminateur [14], ce qui pousse le générateur à produire des données de plus en plus réalistes.
- **L'architecture Transformer** : Fondée sur un mécanisme d'attention, elle est présentée en détail dans la section suivante, car elle constitue la base des grands modèles de langage actuels [3].

1.9 Architecture Transformer

1.9.1 Principe général

Introduit dans l'article *Attention Is All You Need* [3], le Transformer a marqué une rupture importante en apprentissage profond. Contrairement aux architectures récurrentes, il ne traite pas la séquence pas à pas : il s'appuie uniquement sur un mécanisme d'attention, ce qui lui permet de traiter toute la séquence en parallèle.

L'architecture originale suit une structure encodeur-décodeur, initialement conçue pour la traduction automatique. L'encodeur produit une représentation de chaque token ; le décodeur génère ensuite la sortie token par token. Les tokens sont d'abord convertis en vecteurs via une couche d'embedding, à laquelle s'ajoute un encodage positionnel pour tenir compte de l'ordre des mots.

Chaque couche comprend un bloc d'attention multi-têtes et un réseau *feed-forward*, associés à des connexions résiduelles et une normalisation de couche.

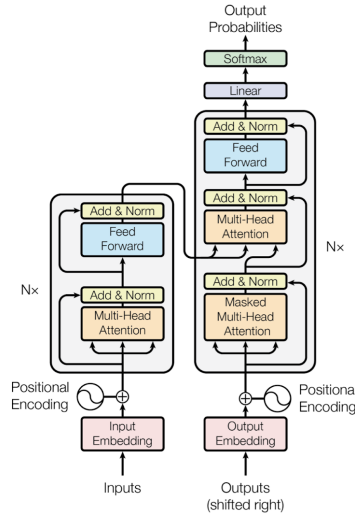


FIG. 1.7 : Architecture générale du Transformer : encodeur (gauche) et décodeur (droite), d'après [3]

1.9.2 Mécanisme d'attention

Le mécanisme d'attention est le cœur du Transformer. Il permet à chaque élément d'une séquence de pondérer l'importance des autres éléments lors du calcul de sa représentation.

- **Attention par produit scalaire mis à l'échelle** : Trois matrices sont dérivées de la séquence d'entrée : les requêtes (Q), les clés (K) et les valeurs (V) [3] :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V \quad (1.1)$$

Le softmax transforme les scores en poids, qui pondèrent ensuite les valeurs V .

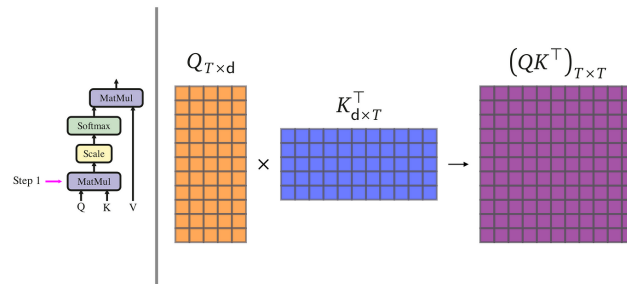


FIG. 1.8 : Schéma du mécanisme Scaled Dot-Product Attention

- **Attention multi-têtes** : Le Transformer calcule plusieurs attentions en parallèle, puis les combine :

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) \cdot W^O \quad (1.2)$$

avec :

$$\text{head}_i = \text{Attention}(Q \cdot W_i^Q, K \cdot W_i^K, V \cdot W_i^V) \quad (1.3)$$

Cela permet au modèle de capturer différents types de relations entre les mots.

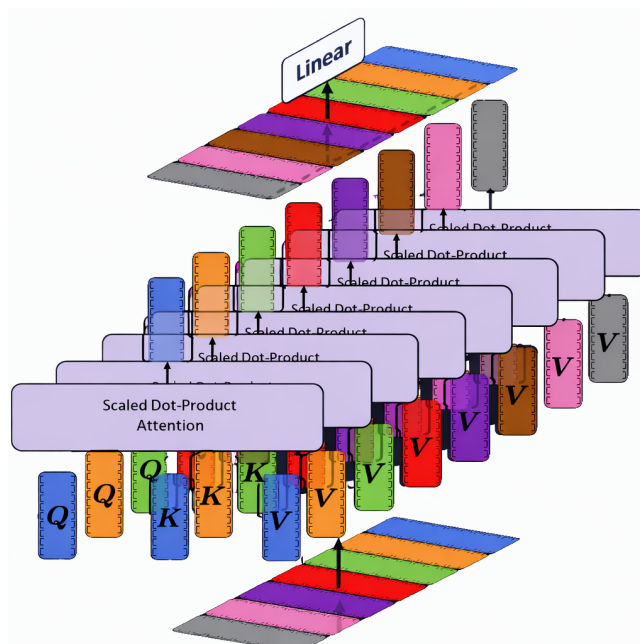


FIG. 1.9 : Schéma de l'attention multi-têtes

- **Auto-attention et attention croisée** : Dans l'encodeur, Q , K et V proviennent de la même séquence (*self-attention*). Dans le décodeur, une attention croisée relie les requêtes du décodeur aux représentations produites par l'encodeur.

1.9.3 Importance dans l'IA moderne

Le Transformer a profondément marqué l'intelligence artificielle depuis sa publication.

Il a d'abord permis d'atteindre de nouveaux niveaux de performance en traitement du langage naturel, dépassant les architectures récurrentes [15][16].

Son fonctionnement en parallèle a aussi facilité l'entraînement de modèles de très grande taille, ouvrant la voie aux grands modèles de langage.

Son usage s'est également étendu au-delà du NLP : le Vision Transformer (ViT) [17] l'a par exemple appliqué à la vision par ordinateur.

Enfin, le Transformer a contribué à l'émergence des modèles de fondation [18] : des modèles pré-entraînés sur de larges corpus, puis adaptés à des tâches spécifiques par fine-tuning ou ingénierie des prompts.

1.10 Grands modèles de langage (LLMs)

Les grands modèles de langage (*Large Language Models*, LLMs) sont des modèles d'IA de très grande échelle, entraînés sur des corpus textuels massifs pour en apprendre les structures statistiques et sémantiques [19].

Un LLM repose sur l'architecture Transformer. Il modélise la probabilité d'une séquence de tokens $(w_1, w_2, \dots, w_n)$ de façon autorégressive :

$$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^n P(w_i \mid w_1, \dots, w_{i-1}) \quad (1.4)$$

À chaque étape, le modèle prédit le token le plus probable en fonction du contexte précédent [16].

Ce qui distingue les LLMs des modèles de langage classiques, c'est leur capacité de généralisation : grâce à leurs milliards de paramètres et à un entraînement à grande échelle, ils développent des connaissances et des capacités de raisonnement qui n'ont pas été explicitement programmées [20]. Ce phénomène, appelé émergence, désigne des capacités qui n'apparaissent qu'au-delà d'un certain seuil de taille du modèle.

1.11 Conclusion

Ce chapitre a posé les fondements théoriques et techniques nécessaires à la compréhension des systèmes intelligents d'aide à la décision. À travers deux axes complémentaires, nous avons d'abord exploré la Business Intelligence, en examinant comment les données brutes sont transformées en connaissances actionnables au service de la prise de décision. Nous avons ensuite abordé les avancées majeures de l'Intelligence Artificielle, notamment les architectures Transformer, les grands modèles de langage (LLMs) et les techniques d'ingénierie des prompts, qui ont révolutionné la manière dont les machines comprennent et génèrent le langage naturel. L'analyse croisée de ces deux domaines met en lumière une convergence naturelle : d'un côté, la BI fournit une infrastructure solide pour la gestion et l'exploitation des données métier ; de l'autre, les LLMs offrent des capacités inédites d'interprétation et de raisonnement sur ces données. Toutefois, ces technologies, prises isolément, présentent des limites — notamment en matière d'accès aux connaissances spécialisées, de fraîcheur de l'information et de fiabilité des réponses générées. C'est précisément pour répondre à ces défis que s'inscrit le chapitre suivant, qui propose une étude conceptuelle du RAG Agentique (Agentic Retrieval-Augmented Generation). Cette approche, en combinant la puissance des LLMs avec une récupération dynamique et contextuelle de l'information, représente une évolution prometteuse vers des systèmes plus fiables, plus autonomes et mieux adaptés aux exigences des applications décisionnelles modernes.

Chapitre 2

Du RAG Classique au RAG Agentique : Fondements et Architecture

Introduction

Les grands modèles de langage (LLMs) ont démontré des capacités remarquables en compréhension et en génération du langage naturel, permettant une large variété d'applications allant des systèmes de questions-réponses ouverts aux systèmes de dialogue spécialisés. Cependant, les LLMs reposent sur des données d'entraînement statiques, ce qui les rend sujets aux hallucinations et limite leur capacité à fournir des informations exactes et à jour dans des tâches dynamiques ou nécessitant des connaissances approfondies.

La génération augmentée par récupération d'information (*Retrieval-Augmented Generation*, RAG) a suscité un intérêt considérable comme approche prometteuse pour surmonter les limitations de connaissances des LLMs qui est un résultat direct du préentraînement statique. Avec une intégration pertinente provenant de bases de connaissances externes ou de moteurs de recherche, le RAG améliore la précision factuelle et élargit la couverture temporelle et thématique du modèle. Les méthodes traditionnelles de RAG ont montré de solides performances lorsque les requêtes sont bien formulées et que les informations nécessaires sont facilement accessibles dans le contexte récupéré.

Malgré l'efficacité des méthodes RAG de base, celles-ci rencontrent souvent des difficultés lorsqu'elles sont appliquées à des applications réelles à grande échelle impliquant des données complexes et hétérogènes. Par exemple, dans les scénarios multi-documents, les informations pertinentes sont réparties entre plusieurs sources, ce qui nécessite non seulement leur récupération, mais également une synthèse cohérente. La simple concaténation des passages récupérés peut conduire à des réponses fragmentées ou contradictoires, en particulier dans des domaines où le raisonnement multi-étapes est essentiel.

Pour remédier à ces limitations des approches RAG classiques dans la gestion de tâches complexes du monde réel, les recherches récentes se sont orientées vers l'**Agentic RAG**, un paradigme qui intègre étroitement la récupération d'information avec le raisonnement et la prise

de décision. Contrairement aux pipelines statiques, l'Agentic RAG considère la récupération non pas comme une étape unique de prétraitement, mais comme une opération dynamique et sensible au contexte, guidée par le processus de raisonnement continu du modèle. Cette perspective centrée sur le raisonnement est essentielle pour les applications nécessitant une résolution de problèmes en plusieurs étapes, une acquisition adaptative d'informations et une synthèse assistée par outils.

Ce chapitre présente une étude conceptuelle de l'Agentic RAG : ses fondements, son architecture, ses composants clés et les défis qu'il soulève.

2.1 Architecture générale d'un système RAG

Pour une requête q , un système RAG typique se compose de trois composants clés, organisés en deux phases distinctes : une phase dédiée à l'indexation, et une phase couvrant la recherche et la génération.

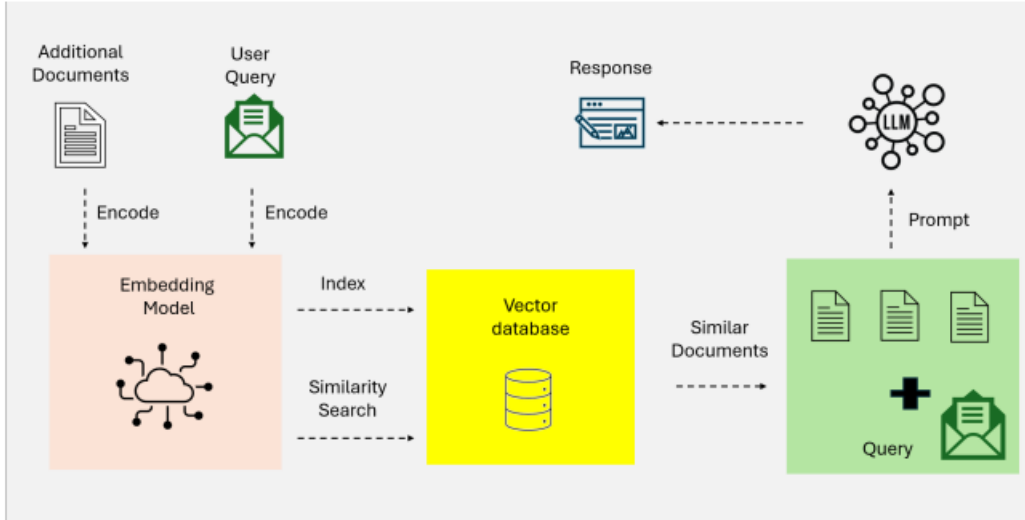


FIG. 2.1 : Architecture générale d'un système RAG, d'après [4]

2.1.1 Indexation (*Indexing*)

L'indexation constitue une phase (*offline*) visant à préparer la base de connaissances avant le traitement des requêtes utilisateur. À partir d'un corpus de documents bruts, une étape de prétraitement appelée *chunking* consiste à découper les documents en fragments de plus petite taille :

$$D = \{d_1, d_2, \dots, d_n\} \quad (2.1)$$

où chaque d_i représente un fragment de document. Ce découpage est une étape non triviale, car la granularité des fragments influence directement la qualité de la récupération.

Chaque fragment d_i est ensuite converti en vecteur dense à l'aide d'un modèle d'embedding $f_e(\cdot)$, puis stocké dans une base de données vectorielle :

$$I = \{e_1, e_2, \dots, e_n\} \quad \text{avec} \quad e_i = f_e(d_i) \in \mathbb{R}^d \quad (2.2)$$

2.1.2 Recherche d'information (*Retrieval*)

La recherche est la première étape de la phase *online*. La requête q est transformée en vecteur à l'aide d'un modèle d'encodage compatible avec celui utilisé lors de l'indexation. Les k fragments les plus similaires à la requête sont ensuite sélectionnés :

$$R : \underset{d_i \in D}{\text{top-}k \text{ Sim}(q, d_i)} \rightarrow D_q \quad (2.3)$$

où $D_q = \{d_1, d_2, \dots, d_k\}$ représente l'ensemble des fragments pertinents pour la requête q . Les fonctions de similarité les plus couramment utilisées sont le produit scalaire et la similarité cosinus :

$$\text{Sim}(q, d_i) = e_q \cdot e_{d_i} \quad \text{ou} \quad \frac{e_q \cdot e_{d_i}}{\|e_q\| \cdot \|e_{d_i}\|} \quad (2.4)$$

À noter que lorsque les vecteurs sont normalisés (L2-normalization), ces deux mesures sont équivalentes.

2.1.3 Génération (*Generation*)

Une fois les fragments pertinents récupérés, la requête q et l'ensemble D_q sont fournis conjointement au LLM via un prompt structuré afin de générer la réponse finale :

$$y = \text{LLM}(\text{Prompt}(D_q, q)) \quad (2.5)$$

où $\text{Prompt}(\cdot)$ désigne un template structuré intégrant une instruction système, les fragments récupérés et la requête. Le LLM synthétise alors ces éléments pour produire une réponse ancrée dans les sources externes.

2.2 Évolution des paradigmes RAG

Les systèmes *Retrieval-Augmented Generation* (RAG) ont évolué afin d'améliorer la qualité des réponses générées par les modèles de langage en combinant recherche d'information et génération de texte. Cette évolution a conduit à plusieurs paradigmes principaux : le **RAG naïf**, le **RAG avancé** et le **RAG modulaire** [21].

2.2.1 RAG Naïf (*Naive RAG*)

Le **RAG Naïf** représente l'architecture classique du RAG. Il repose sur trois étapes principales : l'indexation des documents, la récupération des informations pertinentes à partir d'une requête utilisateur, puis la génération de la réponse par le modèle de langage.

Cette approche permet d'enrichir les réponses du LLM avec des données externes, mais elle présente certaines limites, notamment des problèmes de pertinence dans la récupération des documents et des risques d'hallucinations dans les réponses générées.

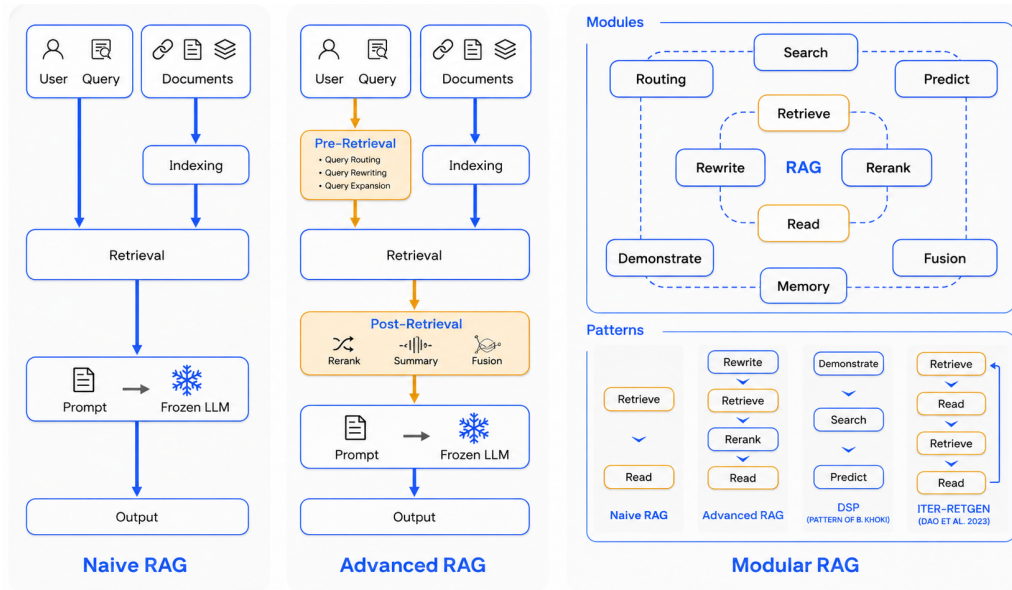


FIG. 2.2 : Évolution des paradigmes RAG : du Naive RAG au Modular RAG, d'après [4]

2.2.2 RAG Avancé (Advanced RAG)

Le **RAG Avancé** introduit plusieurs améliorations afin d'augmenter la qualité de la recherche d'information et la pertinence des réponses. Contrairement au Naive RAG, il intègre des stratégies de *pre-retrieval* et de *post-retrieval*.

Les techniques de *pre-retrieval* permettent d'optimiser la requête ou les documents avant la phase de recherche, tandis que les approches de *post-retrieval* servent à filtrer, réorganiser ou compresser les documents récupérés afin de fournir un contexte plus pertinent au modèle de langage.

Cette approche utilise également des méthodes plus avancées d'indexation et de segmentation des documents afin de réduire les erreurs de récupération et améliorer la qualité des réponses générées. [4]

2.2.3 RAG Modulaire (Modular RAG)

Le **RAG Modulaire** constitue une approche plus flexible et adaptative. Contrairement aux architectures précédentes, il repose sur plusieurs modules spécialisés pouvant être ajoutés ou réorganisés selon les besoins de la tâche.

Cette architecture peut intégrer différents mécanismes comme la mémoire, le routage des requêtes, les recherches hybrides ou encore la reformulation automatique des requêtes. Grâce à cette modularité, le système devient plus performant pour les tâches complexes et les environnements multi-sources. [5]

2.3 Architecture en modules des systèmes RAG modernes

Cette partie présente les principaux modules du cadre RAG modulaire. Dans cette approche, le système est organisé autour de plusieurs étapes essentielles : l'indexation, le pré-retrieval, le retrieval, le post-retrieval et la génération. [5]

2.3.1 Indexation

L'indexation consiste à découper les documents en fragments (*chunks*) afin de faciliter leur stockage et leur recherche dans une base vectorielle. Cette étape joue un rôle fondamental dans la qualité globale du système RAG.

L'un des principaux enjeux concerne la taille des fragments. Des fragments trop grands apportent davantage de contexte mais introduisent également plus de bruit, tandis que des fragments trop petits peuvent manquer d'informations importantes. Pour améliorer cette étape, plusieurs techniques sont utilisées :

- **Sliding Window** : création de fragments qui se chevauchent afin de préserver la continuité sémantique.
- **Ajout de métadonnées** : enrichissement des fragments avec des informations telles que le titre, l'auteur, la page ou la date.
- **Small-to-Big Retrieval** : utilisation de petits fragments pour la recherche et de fragments plus larges pour la génération de réponse.

L'organisation structurelle des documents peut également être améliorée grâce à :

- des index hiérarchiques, organisant les fragments sous forme parent-enfant.
- des graphes de connaissances (*Knowledge Graphs*) permettant de représenter les relations entre entités et concepts.

2.3.2 Pré-retrieval

Le module de pré-retrieval vise à améliorer la requête utilisateur avant la phase de recherche d'information. Cette étape est importante car une requête ambiguë, incomplète ou mal formulée peut réduire la qualité des documents récupérés.

Expansion de requête (*Query Expansion*) L'expansion de requête consiste à enrichir la requête initiale en générant plusieurs variantes ou formulations similaires. Cette approche permet d'augmenter les chances de retrouver des documents pertinents malgré les différences de vocabulaire ou de formulation.

Parmi les méthodes courantes :

- **Multi-Query** : génère plusieurs reformulations d'une même question .
- **Sub-Queries** : décompose une question complexe en plusieurs sous-questions plus simples.

Transformation de requête (*Query Transformation*) La transformation de requête modifie directement la requête utilisateur afin d'améliorer la pertinence de la recherche. L'objectif est de reformuler la question de manière plus claire, plus précise ou plus adaptée au système de retrieval.

Quelques approches fréquemment utilisées :

- **Rewrite** : réécriture automatique de la requête .
- **HyDE** (*Hypothetical Document Embeddings*) : génération d'une réponse hypothétique utilisée pour guider la recherche [22].
- **Step-back Prompting** : reformulation vers une question plus générale afin d'obtenir un contexte plus large.

Construction de requête (*Query Construction*) La construction de requête consiste à convertir une requête en langage naturel vers un format structuré adapté à des bases de données ou graphes de connaissances.

Cette approche permet notamment de générer automatiquement des requêtes :

- SQL pour les bases relationnelles .
- Cypher pour les graphes de connaissances .
- ou d'autres langages spécialisés selon le système utilisé.

2.3.3 Retrieval

Le module de récupération d'information (*retrieval*) constitue le cœur des systèmes RAG, car il permet de rechercher et sélectionner les documents les plus pertinents avant la génération de la réponse. La qualité des informations récupérées influence directement la précision, la pertinence et la fiabilité des réponses produites par le modèle. Plusieurs types de retrievers sont utilisés dans les architectures RAG modernes.

— **Sparse Retriever** : ces méthodes reposent sur des approches statistiques classiques comme TF-IDF ou BM25. Elles fonctionnent principalement à partir des mots-clés présents dans la requête et les documents. Plus un document contient des termes proches de ceux de la requête, plus il est considéré comme pertinent. Ces approches sont simples, rapides et efficaces pour les recherches basées sur des correspondances exactes, mais elles peuvent avoir des difficultés à comprendre le sens global ou les reformulations d'une requête.

— **Dense Retriever** : ces retrievers utilisent des représentations vectorielles appelées *embeddings*, produites par des modèles de langage neuronaux. Les requêtes et les documents sont transformés en vecteurs numériques permettant de mesurer leur proximité sémantique. Cette approche permet de retrouver des documents ayant un sens similaire à la requête, même lorsque les mêmes mots ne sont pas utilisés. Les dense retrievers offrent ainsi une meilleure compréhension du contexte et des intentions de l'utilisateur.

— **Hybrid Retriever** : les approches hybrides combinent les méthodes sparse et dense afin de bénéficier des avantages des deux techniques. Elles associent la précision de la recherche lexicale à la capacité de compréhension sémantique des embeddings. Cette combinaison améliore généralement la qualité de récupération et permet d'obtenir des résultats plus robustes et plus pertinents dans les systèmes RAG modernes.

2.3.4 Post-retrieval

Le module de *post-retrieval* intervient après la récupération des documents afin de nettoyer et optimiser le contexte transmis au modèle de langage. Son objectif est de réduire le bruit informationnel et de ne conserver que les éléments les plus utiles à la génération de la réponse.

- **Reranking** : Réorganise les documents récupérés en fonction de leur pertinence afin de placer en priorité les sources les plus utiles pour la tâche.
- **Compression** : Réduit la taille du contexte en supprimant ou en résumant les informations secondaires afin de respecter la fenêtre de contexte du LLM.
- **Sélection** : Filtre les fragments peu pertinents ou redondants pour ne conserver que les passages les plus informatifs.

2.3.5 Génération

La dernière étape utilise un LLM pour générer une réponse à partir de la requête et des documents sélectionnés. Le modèle exploite les informations contextuelles pour produire une réponse ancrée (*grounded*). Afin de réduire les hallucinations, certains systèmes intègrent une étape de **vérification** pour évaluer la cohérence entre la réponse et les sources.

2.4 Raisonnement à logique prédéfinie dans les systèmes RAG

Les approches de raisonnement à logique prédéfinie reposent sur des *workflows* structurés où les étapes de *retrieval* et de raisonnement suivent une logique de contrôle fixe. Dans ces architectures, les différents modules (reformulation de requête, recherche documentaire, *reranking* et génération) sont organisées selon des *pipelines* déterministes.

L'objectif principal de ces approches est de garantir un contrôle précis du processus de raisonnement tout en conservant une architecture modulaire et réutilisable. Ces *workflows* privilégient

généralement l'efficacité computationnelle, la stabilité et la facilité de personnalisation. Toutefois, leur principal inconvénient réside dans leur dépendance à des chemins d'exécution conçus à l'avance, limitant ainsi leur flexibilité face à des tâches complexes ou évolutives.

Les *workflows* de raisonnement à logique prédéfinie peuvent être regroupés en plusieurs catégories principales : les approches basées sur le routage (*route-based*), les approches itératives (*loop-based*), les approches hiérarchiques (*tree-based*) ainsi que les architectures hybrides modulaires (*hybrid-modular*).

2.4.1 Approches basées sur le routage

Les approches *route-based* utilisent des mécanismes de routage permettant de diriger une requête vers différents *pipelines* de traitement selon des conditions prédéfinies. Le système sélectionne le *workflow* le plus approprié en fonction du contexte, du type de requête ou du niveau de confiance du modèle.

Le routage peut s'appuyer sur plusieurs stratégies :

- **Metadata routing** : exploite des mots-clés, des entités nommées ou des métadonnées structurées extraites de la requête (langue, domaine, type de question) afin de sélectionner les index ou modules les plus pertinents.
- **Semantic routing** : repose sur la comparaison entre la représentation vectorielle de la requête et celles de plusieurs *pipelines*, afin d'identifier l'intention dominante.
- **Hybrid routing** : combine ces deux approches en appliquant d'abord un filtre par métadonnées puis en affinant la sélection par similarité sémantique.

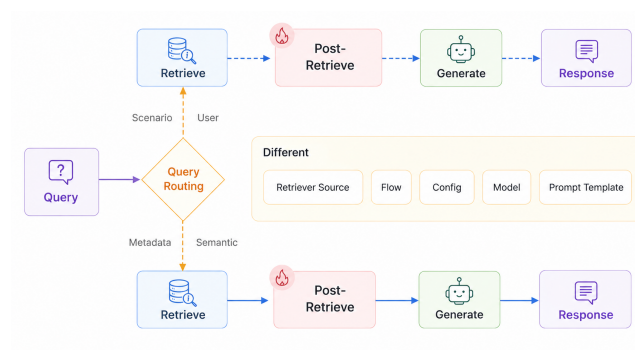


FIG. 2.3 : Schéma des approches basées sur le routage, d'après [5]

Des travaux tels que *RAGate* ou *Self-Route* illustrent cette catégorie en activant le *retrieval* uniquement lorsque des critères paramétrés l'exigent (par exemple, un score de confiance du modèle inférieur à un seuil fixé) ce qui permet de réduire les coûts computationnels.[23]

2.4.2 Approches itératives en boucle

Les approches *loop-based* introduisent des mécanismes de rétroaction permettant au système d'effectuer plusieurs cycles de *retrieval* et de génération. Contrairement aux *pipelines* linéaires, le système peut relancer de nouvelles recherches selon des conditions d'arrêt et des critères de déclenchement fixés à la conception.

Plusieurs variantes peuvent être distinguées :

- **Iterative retrieval** : consiste à enchaîner un nombre borné d'itérations *retrieval/generation*, chaque cycle enrichissant le contexte pour le suivant.
- **Recursive retrieval** : approfondit progressivement la recherche en décomposant dynamiquement la requête en sous-requêtes émergentes à chaque niveau de récursion, permettant de naviguer dans des structures hiérarchiques ou de résoudre des questions *multi-hop*. Un mécanisme de terminaison explicite (tel qu'une profondeur maximale $K_{\max}$) contrôle l'arrêt de la récursion.
- **Retrieval conditionnel** : soumet les documents récupérés à un module d'évaluation externe qui décide, selon des seuils paramétrés, si un nouveau *retrieval* est nécessaire.

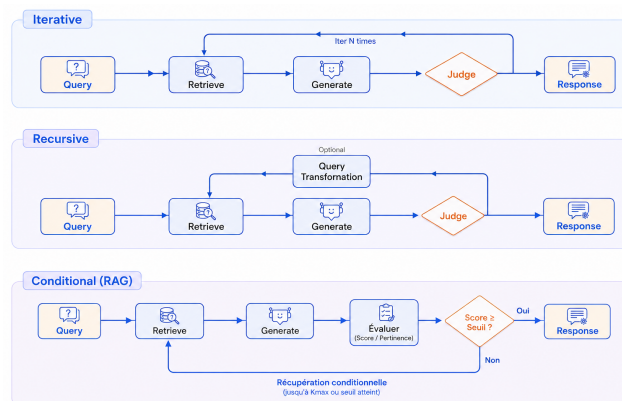


FIG. 2.4 : Schéma des approches itératives en boucle, d'après [5] .

2.4.3 Approches hiérarchiques en arbre

Les approches *tree-based* organisent le *retrieval* et le raisonnement sous forme arborescente, permettant d'explorer différentes branches d'information. La hiérarchie peut porter sur les documents eux-mêmes ou sur la décomposition de la question initiale.

Le système construit généralement des arbres de résumés ou de sous-questions afin de naviguer efficacement dans de grands corpus :

- *RAPTOR* exploite cette logique en construisant récursivement des arbres de résumés à plusieurs niveaux d'abstraction.[24]

2.4.4 Approches hybrides modulaires

Les architectures *hybrid-modular* représentent la forme la plus avancée des *workflows* prédéfinis. Elles combinent au sein d'un même système des mécanismes de routage conditionnel, de boucles itératives et de modules spécialisés.[5]

Leur fonctionnement repose sur trois mécanismes essentiels :

1. **Routing** : sélectionne les modules appropriés selon des règles prédéfinies.
2. **Scheduling** : contrôle l'ordre d'exécution des opérations et les conditions de déclenchement.
3. **Fusion** : agrège les informations provenant de plusieurs branches via des techniques telles que le *Reciprocal Rank Fusion* (RRF), les ensembles pondérés (*weighted ensemble*) ou la synthèse par LLM.

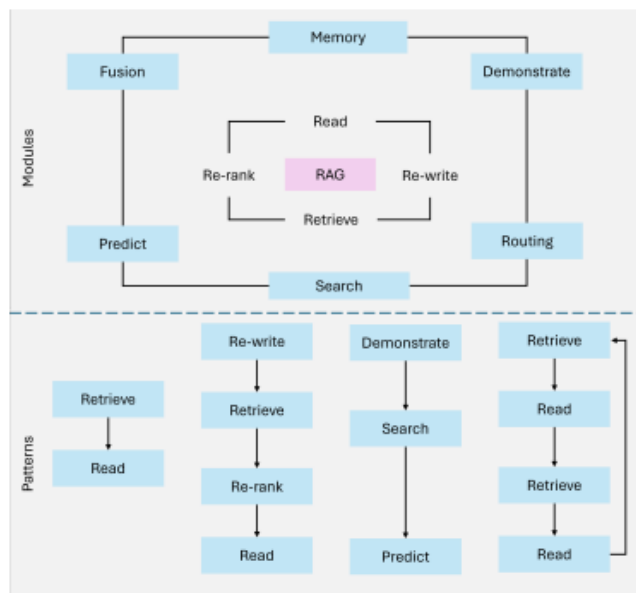


FIG. 2.5 : Schéma des approches hybrides modulaires, d'après [4]

2.4.5 Limites structurelles des pipelines RAG prédéfinis

Les architectures RAG prédéfinies dissocient la récupération des connaissances de la génération de texte à travers un mécanisme déterministe. Même dans leurs formes les plus sophistiquées, ces architectures se heurtent à trois limitations structurelles :

- **Sensibilité à la qualité du *retrieval* initial.** La qualité de la réponse finale dépend fortement de la pertinence des documents récupérés. Si l' *embedding* associe la requête à des documents sous-optimaux, le générateur est contraint de produire sa réponse à partir d'informations bruitées. À l'inverse, un contexte trop volumineux et peu discriminant peut dégrader la qualité du raisonnement du modèle par surcharge contextuelle (*contextual overhead*).

- **Rigidité face aux tâches non anticipées.** La logique du *pipeline* étant fixée à la conception, le système ne peut pas reformuler son plan de résolution face à une requête dont la structure diffère des cas prévus. Lorsqu'une question complexe requiert de synthétiser des informations provenant de sources ne partageant pas de forte similarité sémantique, ou lorsque la nature du problème se révèle différente de ce qu'anticipait le *pipeline*, aucun mécanisme ne permet de réorienter le traitement en cours d'exécution.
- **Propagation des erreurs sans correction autonome.** La séparation stricte entre *retrieval* et génération empêche le modèle d'interrompre sa production pour identifier une lacune informationnelle et la combler. Les erreurs introduites lors du *retrieval* se propagent donc jusqu'à la génération finale, conduisant fréquemment à des hallucinations lorsque le contexte récupéré est insuffisant ou incomplet.

2.5 Raisonnement agentique (*Agentic Reasoning*)

Contrairement aux *workflows* prédéfinis, le raisonnement agentique considère le LLM comme un agent autonome capable de prendre des décisions dynamiques durant le processus de raisonnement. Le modèle peut identifier des lacunes informationnelles, décider quand lancer une recherche externe, sélectionner des outils, reformuler des requêtes ou planifier plusieurs étapes de résolution. Le *retrieval* n'est donc plus une étape fixe du *pipeline* mais une action déclenchée de manière autonome selon les besoins du raisonnement.

Le raisonnement agentique repositionne le LLM comme un agent décisionnel actif, capable d'orchestrer de manière autonome les opérations de récupération d'information et l'utilisation d'outils tout au long du processus de raisonnement. Au lieu d'exécuter un plan fixe, le modèle identifie ses lacunes en connaissances, formule des requêtes, récupère des informations externes via des outils tels que des moteurs de recherche ou des *APIs*, puis intègre ces informations dans une solution en constante évolution.[23]

Cette interaction dynamique entre raisonnement et utilisation d'outils permet au système de traiter des tâches complexes nécessitant plusieurs étapes, ainsi qu'un raffinement itératif et une synthèse adaptative de l'information. Il existe deux principales approches pour mettre en œuvre le raisonnement agentique :

- **Méthodes basées sur le *prompt*** : Dans ce cas, le modèle exploite ses capacités de raisonnement en contexte (*in-context reasoning*) et de suivi d'instructions pour décider de ses actions. Il est guidé par des consignes soigneusement formulées ou des *tokens* de contrôle intégrés qui indiquent quand effectuer une récupération, quelles actions entreprendre et comment intégrer les informations externes. Cette approche ne nécessite aucun entraînement supplémentaire, ce qui la rend légère et facilement adaptable à différents types de tâches.
- **Méthodes basées sur l'entraînement** : Ici, les modèles sont explicitement optimisés pour apprendre quand et comment utiliser des outils externes. Cette optimisation permet un usage plus stratégique et plus précis des outils, ainsi que l'apprentissage de capacités de planification à long terme et de politiques de récupération (*retrieval policies*) adaptées à des tâches complexes.

TAB. 2.1 : Comparaison des flux de raisonnement (*Reasoning Workflows*)

Raisonnement	<i>Flux de travail</i>	Description
Système 1	<i>Raisonnement prédéfini</i>	Exécution structurée, modulaire et basée sur des règles.
Système 2	<i>Raisonnement agentique</i>	Prise de décision autonome, adaptative et pilotée par le modèle.

2.6 Formalisation générale des systèmes agentiques basés sur les LLM

Afin de modéliser le fonctionnement général des systèmes agentiques fondés sur les grands modèles de langage (LLM), on peut représenter le système par le tuple suivant :

$$S_{Agent} = \langle I, S, A, \Omega, O, \Pi, M, \Psi \rangle \quad (7)$$

Où les composants sont définis comme suit :

Paramètre	Description
I	Ensemble des agents composant le système.
S	Espace des états de l'environnement.
A	Ensemble des actions possibles.
Ω	Espace des observations accessibles aux agents.
$O(o_t^i \mid s_t, h_t^i, Q)$	Fonction d'observation (vue partielle de l'environnement).
$\Pi = \{\pi_i\}_{i \in I}$	Ensemble des politiques de décision des agents.
M	Ensemble des mécanismes de mémoire utilisés par les agents.
$\Psi(s_{t+1} \mid s_t, a_t)$	Fonction de transition décrivant l'évolution de l'environnement.

TAB. 2.2 : Composants du tuple des systèmes agentiques basés sur les LLM

Afin de mieux comprendre le fonctionnement global du système, chaque composant du tuple peut être interprété comme un élément spécifique du processus décisionnel et du mécanisme d'interaction des agents [25].

2.6.1 Ensemble des agents (I)

Le terme $I = \{1, \dots, N\}$ représente l'ensemble des agents présents dans le système. Lorsque $N = 1$, le système correspond à un agent unique. Lorsque $N > 1$, le système devient multi-agent et peut inclure plusieurs agents spécialisés collaborant autour d'un objectif commun.

Chaque agent peut posséder :

- un rôle spécifique .
- une mémoire propre .
- une politique de décision indépendante .
- des capacités d'interaction avec les autres agents.

2.6.2 État de l'environnement (S)

L'espace S représente l'ensemble des états possibles de l'environnement. Cet environnement peut inclure des bases de connaissances, des documents, des outils externes, un dépôt logiciel, un navigateur web, un espace mémoire partagé ou un environnement physique. L'état réel de l'environnement est généralement partiellement observable : les agents n'ont accès qu'à une vue limitée à travers leurs observations.

2.6.3 Espace des actions (A)

L'ensemble A définit toutes les actions exécutables. Contrairement aux systèmes conversationnels classiques, les systèmes agentiques disposent d'un espace d'actions riche :

- génération de texte ou d'étapes de raisonnement ;
- appel à des outils externes et exécution de code ;
- recherche documentaire et planification de sous-tâches ;
- mise à jour d'une mémoire et communication inter-agents ;
- modification de l'environnement ou arrêt du processus (STOP).

2.6.4 Espace des observations (Ω)

L'espace Ω correspond aux informations perceptibles par les agents : documents récupérés, résultats d'APIs, sorties d'outils, messages inter-agents ou traces de raisonnement. Ces observations enrichissent progressivement le contexte de décision.

2.6.5 Fonction d'observation ($O(o_t^i \mid s_t, h_t^i, Q)$)

Elle décrit la perception de l'agent i à l'instant t . L'observation dépend de l'état courant (s_t), de l'historique visible (h_t^i) et de la tâche globale (Q). L'historique peut contenir les échanges précédents, les sorties d'outils et les contributions des pairs, modélisant ainsi la visibilité partielle de l'environnement.

2.6.6 Politiques de décision (Π)

L'ensemble $\Pi = \{\pi_i\}_{i \in I}$ représente les politiques décisionnelles. Pour un agent i , la politique (souvent un LLM) sélectionne l'action :

$$a_t = \pi_i(o_t^i, m_t^i, Q) \quad (2.6)$$

où o_t^i est l'observation, m_t^i la mémoire et Q la tâche. La politique orchestre le choix des outils, la planification et la coordination.

2.6.7 Mémoire des agents (M)

Le composant M regroupe les mécanismes de stockage (documents, plans, résumés). Les systèmes modernes emploient des stratégies avancées : compression, résumé automatique, suppression des redondances et gestion de la mémoire à long terme ou partagée.

2.6.8 Fonction de transition ($\Psi(s_{t+1} \mid s_t, a_t)$)

Elle décrit l'évolution de l'environnement après une action. La dynamique est cyclique : observation $\rightarrow$ sélection d'action $\rightarrow$ exécution $\rightarrow$ évolution de l'état $\rightarrow$ nouvelles observations. Ce cycle se poursuit jusqu'à satisfaction des conditions d'arrêt.

2.7 Propriétés des systèmes *Agentic RAG*

À partir de cette formalisation, plusieurs propriétés essentielles permettent de caractériser un système *Agentic RAG* :

- 1) **Contrôle itératif** : Le système doit disposer d'une boucle de rétroaction lui permettant d'effectuer plusieurs cycles de raisonnement et de *retrieval* avant la génération finale. Cette boucle est pilotée par une politique de contrôle dynamique qui décide continuellement des prochaines actions à effectuer selon l'évolution du contexte.
- 2) **Retrieval dynamique** : Les requêtes de recherche ne sont pas prédéfinies de manière statique. Elles sont générées dynamiquement au cours de l'exécution en fonction de l'état courant de la mémoire et des besoins informationnels identifiés durant le raisonnement.
- 3) **Actionnabilité des outils externes** : Le *retriever* ainsi que les outils externes doivent être considérés comme des actions explicites du système. L'agent peut ainsi invoquer des moteurs de recherche, *APIs* ou bases de connaissances comme composants intégrés au processus de raisonnement.
- 4) **Persistance de la mémoire** : Le système doit maintenir une mémoire de travail permettant de conserver les informations accumulées au fil des différentes étapes du raisonnement. Cette mémoire contient notamment les documents récupérés, les observations, les conclusions intermédiaires ainsi que l'historique des interactions.

2.8 Taxonomie de l'Agentic RAG

La génération augmentée par récupération (RAG) associe un mécanisme de recherche d'information (*Retriever*) à un générateur afin d'ancrer les réponses du modèle dans des connaissances externes plutôt que de se baser uniquement sur ses connaissances internes.

L'*Agentic RAG* étend ce paradigme en introduisant explicitement un planificateur (*Planner*) qui contrôle l'invocation des outils, y compris la récupération d'informations, selon une politique de contrôle. Cela permet la récupération itérative, la construction dynamique du contexte et le raisonnement multi-étapes au-delà d'un simple schéma « rechercher puis générer ».

Cette section présente une typologie fondée sur plusieurs dimensions orthogonales permettant de classer les systèmes RAG agentiques selon leur mode de fonctionnement plutôt que selon leur implémentation technique. Les systèmes sont ainsi classés selon quatre dimensions principales :

- L'architecture du système .
- La stratégie de récupération .
- Le paradigme de raisonnement .
- La gestion de la mémoire et du contexte.

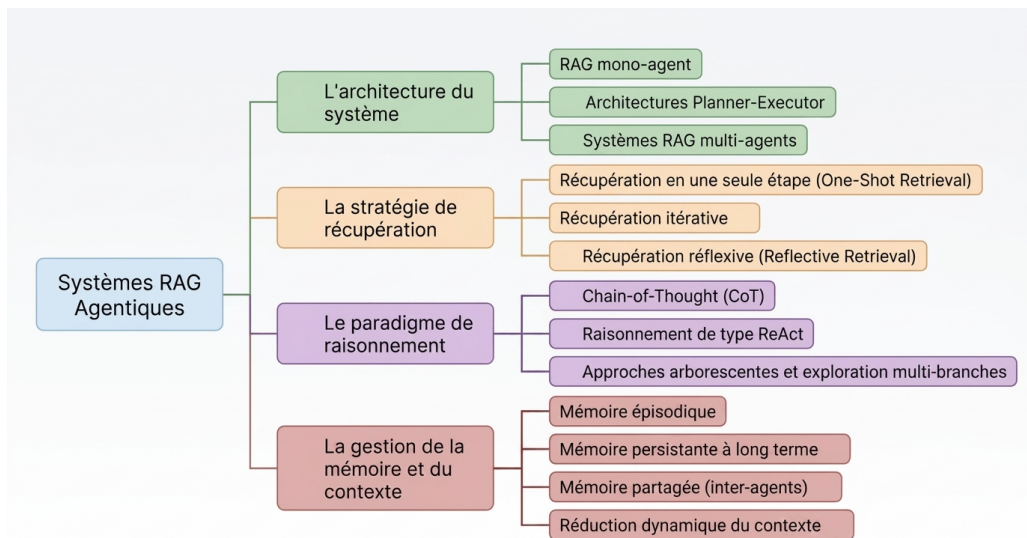


FIG. 2.6 : Taxonomie de l'Agentic RAG, d'après[6].

2.8.1 Taxonomie architecturale

La typologie architecturale classe les systèmes selon leur topologie d'agents, c'est-à-dire le nombre d'entités décisionnelles, l'emplacement du planificateur et la distribution des rôles.

- **RAG mono-agent** : Un seul agent assure à la fois la planification et la génération, tout en invoquant les mécanismes de récupération sous une politique de contrôle unique.

- **Architectures *Planner–Executor*** : Elles séparent le planificateur, chargé de décomposer les objectifs, de l'exécuteur, responsable d'effectuer les récupérations et de retourner les observations.
- **Systèmes RAG multi-agents** : Distribuent la planification, la récupération et la génération entre plusieurs agents collaborant via des interactions distribuées.

2.8.2 Taxonomie des stratégies de récupération

Cette dimension décrit quand et comment le *retriever* est invoqué au cours d'une trajectoire de résolution.

- **Récupération en une seule étape (*One-Shot Retrieval*)** : Une unique opération de récupération initiale suivie d'une génération. C'est le fonctionnement traditionnel du RAG.
- **Récupération itérative** : Réalise plusieurs opérations de récupération où les recherches ultérieures dépendent de l'état intermédiaire et des raisonnements produits.
- **Récupération réflexive (*Reflective Retrieval*)** : Combine récupération, critique et révision. Le système évalue ses propres générations pour affiner progressivement ses requêtes.

2.8.3 Taxonomie du raisonnement

Classe les formes de raisonnement multi-étapes utilisées pour transformer les preuves récupérées en réponses fondées.

- ***Chain-of-Thought* (CoT)** : Le raisonnement de type *Chain-of-Thought* consiste à décomposer explicitement la résolution d'un problème en une suite d'étapes intermédiaires avant la génération de la réponse finale. Le modèle produit ainsi une trace de raisonnement séquentielle permettant de structurer l'inférence et de réduire les erreurs sur les tâches complexes. Dans les systèmes RAG, cette approche peut être utilisée pour guider la formulation des requêtes, organiser l'exploitation des documents récupérés ou justifier progressivement la réponse générée. Toutefois, le flot de raisonnement reste généralement linéaire et suit une trajectoire unique sans mécanisme explicite d'exploration alternative.[26]
- **Raisonnement de type *ReAct*** : Le paradigme *ReAct* (*Reasoning and Acting*) combine explicitement raisonnement symbolique et exécution d'actions externes. Le modèle alterne entre des étapes de réflexion (*Thought*) et des actions (*Action*) telles que des appels d'outils, des requêtes de retrieval ou des consultations de bases de connaissances. Cette approche permet au système d'adapter dynamiquement sa stratégie en fonction des informations obtenues au cours de l'exécution. Contrairement au *CoT* classique, le raisonnement n'est plus uniquement interne et linéaire, mais devient interactif et orienté vers la prise de décision conditionnelle.[27]

- **Approches arborescentes et exploration multi-branches** : Les approches de type *Tree-of-Thought* modélisent le raisonnement comme une exploration d'un arbre de solutions candidates plutôt qu'une unique chaîne séquentielle. Chaque nœud représente un état intermédiaire du raisonnement et plusieurs branches peuvent être explorées simultanément avant sélection de la trajectoire la plus pertinente. Ces architectures introduisent des mécanismes d'évaluation intermédiaire, de comparaison entre branches et de retour arrière (*backtracking*) afin de corriger les trajectoires peu prometteuses. Dans ce contexte, des systèmes tels que *MCTS-RAG* exploitent des mécanismes inspirés du *Monte Carlo Tree Search* (MCTS) pour guider l'exploration de l'espace de raisonnement. Le système sélectionne dynamiquement les branches à développer selon une politique équilibrant exploration de nouvelles pistes et exploitation des trajectoires jugées les plus pertinentes, améliorant ainsi la robustesse du raisonnement et la qualité des réponses générées.[28]

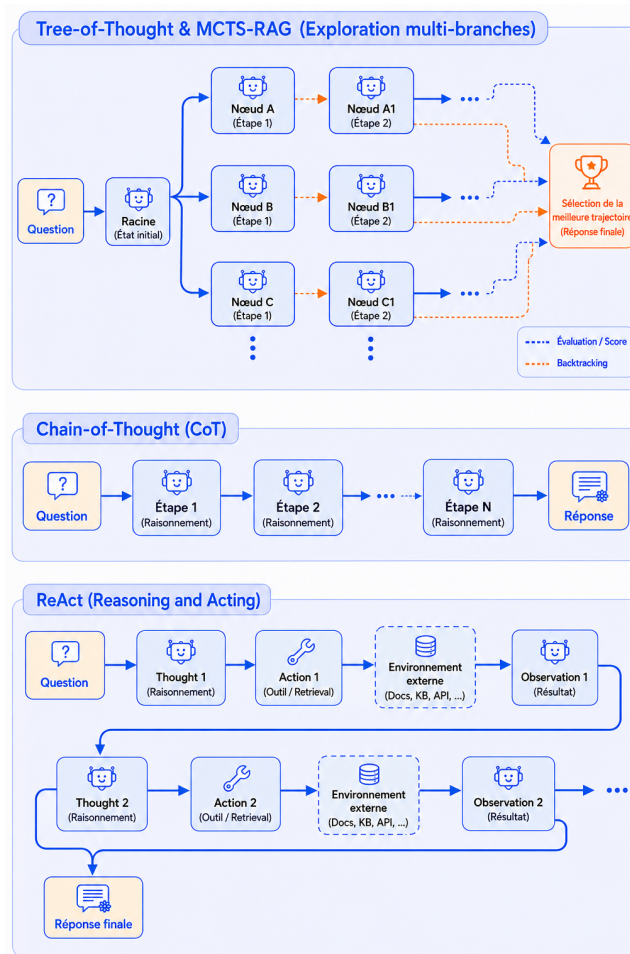


FIG. 2.7 : Taxonomie du raisonnement

2.8.4 Paradigmes de mémoire et de contexte

Les systèmes doivent gérer la mémoire persistante et le contexte actif (*working memory*). Des techniques de réduction dynamique du contexte (*context compression*) permettent de filtrer les informations inutiles. On distingue :

- **Mémoire épisodique** : Stocke les trajectoires temporaires et les rétroactions pour préserver la cohérence locale.
- **Mémoire persistante à long terme** : Conserve des informations entre les sessions, transformant l'architecture d'un fonctionnement *stateless* vers une entité continue.
- **Mémoire partagée (inter-agents)** Dans un système multi-agents, une zone de mémoire accessible à plusieurs agents simultanément. Un agent écrit ses résultats, un autre les lit et les complète. C'est typiquement implémenté via une base de données partagée ou un bus de messages.

2.8.5 Choix architecturaux et interactions entre les dimensions

Un système Agentique RAG peut être vu comme le résultat d'une combinaison de décisions prises sur chacun des axes présentés dans la partie précédente. Chaque architecture correspond ainsi à une configuration particulière définissant la manière dont sont organisés le raisonnement, la mémoire, l'utilisation des outils et les mécanismes de récupération d'information. Il n'existe donc pas de solution unique capable d'optimiser simultanément tous les aspects du système ; les choix effectués dépendent principalement des objectifs visés, des contraintes techniques et du niveau de complexité recherché.

La conception d'un système RAG agentique consiste donc à définir une stratégie particulière pour chacun des quatre axes. Différentes combinaisons architecturales peuvent ainsi être obtenues :

- Une architecture peut combiner un raisonnement de type *ReAct*, une mémoire conversationnelle à court terme, et une récupération itérative permettant d'affiner progressivement les documents récupérés.
- Un autre système peut adopter une organisation multi-agents avec planification hiérarchique, une mémoire persistante partagée entre agents et une stratégie de récupération auto-raffinée basée sur la reformulation successive des requêtes.
- À l'inverse, une architecture plus légère peut reposer sur un agent unique, une mémoire minimale limitée au contexte courant et une récupération *one-shot* afin de réduire la latence et les coûts computationnels.

En pratique, ces différentes dimensions ne fonctionnent toutefois pas de manière isolée. Les choix réalisés sur un axe influencent directement les autres composantes du système. Par exemple, l'introduction d'une mémoire plus riche ou d'une orchestration multi-agents améliore souvent les capacités de raisonnement et de collaboration, mais augmente également les coûts de coordination, la latence et la complexité globale du système. La conception d'un Agentique RAG repose ainsi sur un ensemble de compromis entre performance, coût, robustesse et simplicité architecturale [6].

2.9 Évaluation des systèmes RAG classiques

Les systèmes RAG classiques sont évalués à l'aide de métriques issues de deux champs distincts : la génération de texte et la recherche d'information.

- **BLEU** (*Bilingual Evaluation Understudy*, Papineni et al., 2002) mesure le chevauchement des n -grammes entre la réponse générée et une ou plusieurs références [29] :

$$\text{BLEU} = BP \cdot \exp \left(\sum_{n=1}^N w_n \log p_n \right) \quad (2.7)$$

où p_n est la précision modifiée des n -grammes, w_n les pondérations associées, et BP la pénalité de brièveté (*brevity penalty*). Initialement conçu pour la traduction automatique, BLEU capture uniquement la similarité lexicale de surface.

- **ROUGE** (*Recall-Oriented Understudy for Gisting Evaluation*, Lin, 2004) mesure le recouvrement entre la réponse générée et les références [30]. La variante ROUGE-N est définie par :

$$\text{ROUGE-N} = \frac{\sum_{S \in \text{références}} \sum_{gram_n \in S} \text{Count}_{\text{match}}(gram_n)}{\sum_{S \in \text{références}} \sum_{gram_n \in S} \text{Count}(gram_n)} \quad (2.8)$$

Pour pallier les limites lexicales, des métriques orientées récupération sont nécessaires. Le **Recall@k** mesure la capacité du système à retrouver les documents pertinents parmi les k premiers résultats :

$$\text{Recall@k} = \frac{\text{Nombre de documents pertinents retrouvés parmi les } k \text{ premiers}}{\text{Nombre total de documents pertinents}} \quad (2.9)$$

Des frameworks intégrés comme **RAGAS** (Es et al., 2023) permettent d'évaluer simultanément la fidélité (*Faithfulness*), la pertinence de la réponse et celle du contexte.

2.10 Limites des métriques classiques face aux systèmes Agentic RAG

Les systèmes *Agentic RAG* introduisent des mécanismes de raisonnement multi-étapes et des appels d'outils dynamiques. Les métriques traditionnelles deviennent structurellement insuffisantes pour trois raisons principales :

1. Elles comparent une sortie finale à une référence statique, ignorant la **trajectoire entière** (planification, décisions intermédiaires).
2. Elles ne distinguent pas une réponse correcte obtenue par un raisonnement valide d'une réponse obtenue par une **chaîne d'inférences erronée**.
3. Les métriques de récupération classiques évaluent une action ponctuelle, alors que les agents réalisent des **récupérations successives et conditionnelles**.

2.11 Pipeline d'évaluation structuré des systèmes Agentic RAG

Les systèmes Agentic RAG combinant raisonnement itératif, mémoire et interaction avec des outils, leur évaluation doit être réalisée à plusieurs niveaux complémentaires. Une approche structurée peut être organisée en trois couches principales.

2.11.1 Évaluation au niveau des composants

Cette première couche évalue individuellement chaque module du système afin d'identifier les erreurs locales avant leur propagation dans la chaîne de raisonnement. Elle concerne notamment :

- le planificateur .
- le moteur de récupération .
- les modules d'exécution d'outils .
- les systèmes mémoire.

L'objectif est d'isoler les défaillances spécifiques sans les confondre avec les erreurs produites ultérieurement.

2.11.1.0.1 Planificateur : Goal Decomposition Accuracy mesure la capacité du planificateur à couvrir l'ensemble des sous-tâches attendues lors de la décomposition d'un objectif complexe :

$$DR = \frac{1}{N} \sum_{i=1}^N \frac{|S_i^{\text{pred}} \cap S_i^{\text{ref}}|}{|S_i^{\text{ref}}|} \quad (2.10)$$

où :

- S_i^{pred} est l'ensemble des sous-tâches générées par le planificateur pour la requête i .
- S_i^{ref} est l'ensemble des sous-tâches de référence (*ground truth*) .
- N est le nombre total de requêtes évaluées.

Un score de 1 indique une couverture parfaite des sous-tâches de référence. Cette métrique quantifie dans quelle mesure le planificateur identifie toutes les sous-tâches nécessaires, sans pénaliser les sous-tâches superflues éventuellement générées en surplus. Pour une évaluation symétrique, elle peut être complétée par une mesure de précision ou combinée en F1.

2.11.1.0.2 Moteur de récupération : Le **MRR** évalue la position du premier document pertinent récupéré, en calculant la moyenne des rangs réciproques sur l'ensemble des requêtes :

$$MRR = \frac{1}{N} \sum_{i=1}^N \frac{1}{\text{rank}_i} \quad (2.11)$$

où rank_i désigne la position du premier document pertinent retourné pour la requête i . Un MRR de 1 signifie que le document pertinent est systématiquement en première position.

Le **nDCG@k** prend en compte à la fois la pertinence graduelle des documents et leur position dans la liste retournée. Il se calcule via le **DCG@k**, qui somme les scores de pertinence en appliquant une pondération logarithmique décroissante :

$$\text{DCG@k} = \sum_{i=1}^k \frac{\text{rel}_i}{\log_2(i+1)} \quad (2.12)$$

où rel_i est le score de pertinence du document en position i . Le nDCG@k normalise ensuite ce score par le DCG idéal (IDCG@k), calculé sur un classement parfait :

$$\text{nDCG@k} = \frac{\text{DCG@k}}{\text{IDCG@k}} \quad (2.13)$$

Un nDCG@k égal à 1 correspond à un classement parfait.

2.11.1.0.3 Outils externes : Le **Tool Call Success Rate (TCSR)** mesure la fiabilité des appels aux outils externes sur l'ensemble des requêtes du système :

$$\text{TCSR} = \frac{1}{N} \sum_{i=1}^N \frac{c_i^{\text{success}}}{c_i^{\text{total}}} \quad (2.14)$$

où :

- c_i^{success} est le nombre d'appels d'outils réussis pour la requête i .
- c_i^{total} est le nombre total d'appels d'outils effectués pour la requête i .
- N est le nombre total de requêtes évaluées.

Un appel est considéré réussi lorsqu'il retourne une réponse valide, exploitable par l'agent sans déclenchement de mécanisme de repli.

2.11.2 Évaluation au niveau des trajectoires

Cette couche analyse la manière dont les actions successives s'enchaînent pour former une trajectoire de raisonnement cohérente.

- **Progress Rate (PR)** : mesure la proportion d'étapes de raisonnement contribuant effectivement à l'avancement de la tâche.

$$\text{PR} = \frac{\text{Nombre d'étapes utiles}}{\text{Nombre total d'étapes}} \quad (2.15)$$

Une étape est considérée utile si elle réduit l'écart entre l'état courant et l'objectif final, ce qui peut être évalué par vérification post-hoc ou par un juge LLM.

- **Task Completion Rate (TCR)** : mesure le taux de réussite globale sur l'ensemble des trajectoires évaluées.

$$\text{TCR} = \frac{1}{N} \sum_{i=1}^N \mathbf{1}[\text{tâche}_i \text{ réussie}] \quad (2.16)$$

où $\mathbf{1}[\cdot]$ est la fonction indicatrice valant 1 si la tâche i a été menée à bien selon un critère de succès prédéfini, et 0 sinon.

2.11.3 Évaluation au niveau système

Cette dernière couche considère le système Agentique RAG dans sa globalité. L'évaluation porte alors sur la réussite finale des tâches, la coordination entre agents, la fidélité globale des réponses ainsi que les contraintes pratiques de déploiement.

Des aspects opérationnels critiques incluent :

- **La consommation moyenne de tokens (Token Usage)** sur un ensemble de N requêtes :

$$\overline{\text{TokenUsage}} = \frac{1}{N} \sum_{i=1}^N (N_{\text{input},i} + N_{\text{output},i} + N_{\text{retrieval},i})$$

où :

- * $N_{\text{input},i}$ représente le nombre de tokens d'entrée pour la requête i .
- * $N_{\text{output},i}$ le nombre de tokens générés.
- * $N_{\text{retrieval},i}$ les tokens issus des documents récupérés ou des appels d'outils.

- **La latence moyenne de bout en bout (E2E Latency)** :

$$\overline{\text{Latency}_{\text{E2E}}} = \frac{1}{N} \sum_{i=1}^N (t_{\text{final},i} - t_{\text{start},i})$$

où :

- * $t_{\text{start},i}$ correspond au début de l'exécution de la requête i .
- * $t_{\text{final},i}$ à la génération complète de la réponse.

- **Le délai moyen avant génération du premier token (TTFT – Time To First Token)** :

$$\overline{\text{TTFT}} = \frac{1}{N} \sum_{i=1}^N (t_{\text{first_token},i} - t_{\text{start},i})$$

où $t_{\text{first_token},i}$ représente l'instant de génération du premier token pour la requête i .

Ces éléments deviennent des critères importants pour évaluer la viabilité opérationnelle du système dans des conditions réelles de déploiement. Ces métriques sont particulièrement critiques dans les systèmes Agentique RAG, car les boucles itératives de raisonnement, les appels multiples aux *retrievers* et l'utilisation d'outils externes peuvent fortement augmenter les coûts et les temps d'exécution par rapport aux pipelines RAG classiques.

2.12 Limites, modes de défaillance et défis de fiabilité

Les sections précédentes ont présenté les architectures et les principaux paradigmes des systèmes *Agentic RAG*. Cette partie met en évidence leurs limites structurelles ainsi que leurs vulnérabilités systémiques. Le passage des *pipelines* statiques de type *retrieve-then-generate* à des *workflows* agentiques multi-étapes intégrant des outils introduit de nouvelles surfaces de défaillance. Contrairement aux systèmes RAG traditionnels, les erreurs dans ces architectures ne restent pas isolées : elles sont itératives et peuvent s'amplifier au cours du processus de raisonnement.

2.12.0.1 Dérive de récupération et désalignement des requêtes

Dans les systèmes RAG classiques, la qualité de la récupération dépend essentiellement de la requête initiale. Dans les systèmes agentiques, les requêtes sont reformulées de manière itérative par l'agent, ce qui peut entraîner une dérive sémantique (*semantic drift*) progressive entre la requête évolutive et le besoin initial de l'utilisateur.

Cette problématique apparaît notamment dans les architectures multi-agents :

- un agent planificateur peut mal interpréter les sous-objectifs à déléguer .
- les agents de récupération peuvent recevoir des requêtes ambiguës ou incomplètes .
- le système peut progressivement s'éloigner de l'intention réelle de l'utilisateur.

En l'absence de mécanismes explicites de convergence ou de retour de qualité, ces processus deviennent instables et peuvent consommer des ressources importantes sans produire de réponse satisfaisante.

2.12.0.2 Mauvaise utilisation des outils et erreurs en cascade

Les systèmes *Agentic RAG* étendent les capacités des modèles de langage à l'utilisation d'outils externes tels que les *APIs*, les bases de données ou l'exécution de code. Chaque interaction constitue un point potentiel de défaillance. Le modèle peut :

- sélectionner un outil inadapté .
- générer une requête (*query*) incorrecte .
- mal interpréter les résultats retournés .
- rencontrer des erreurs d'exécution ou des délais de réponse (*timeouts*).

Dans les *workflows* multi-étapes, ces erreurs peuvent être propagées au reste du raisonnement. Une sortie erronée peut être considérée comme valide et servir de base aux étapes suivantes, créant ainsi des erreurs en cascade.

2.12.0.3 Injection de prompt

Les systèmes qui récupèrent des informations depuis des sources ouvertes sont vulnérables aux attaques par injection de *prompt* (*prompt injection*). Des instructions malveillantes peuvent être intégrées dans les documents récupérés afin de :

- modifier la planification de l'agent .
- influencer les décisions de raisonnement .
- provoquer l'utilisation d'outils non prévus .
- détourner le comportement global du système.

Chaque nouvelle étape de récupération constitue un nouveau point d'exposition, rendant la distinction entre instructions légitimes et contenus adversariaux complexe.

Ainsi, les risques dans les systèmes agentiques ne proviennent plus uniquement d'erreurs isolées de génération, mais de dynamiques itératives complexes impliquant récupération, mémoire et utilisation d'outils.

Chapitre 3

Conception et Réalisation de la Solution

3.1 Introduction

Ce chapitre est consacré à la présentation détaillée de la solution proposée dans le cadre de ce projet. Nous y décrivons l'architecture globale du système ainsi que les différents composants qui interviennent dans le traitement des requêtes analytiques formulées en langage naturel. L'objectif principal de cette solution est de permettre l'interrogation intelligente de sources de données hétérogènes à travers une architecture agentique modulaire, capable de générer des analyses cohérentes et contextualisées.

Nous présenterons d'abord la vue d'ensemble du système ainsi que les choix architecturaux ayant conduit à l'adoption d'une approche agentique décentralisée. Nous détaillerons ensuite les principaux modules de l'architecture. Ce chapitre abordera également le workflow complet du système, depuis l'analyse de l'intention utilisateur jusqu'à la génération finale des réponses analytiques et des visualisations. Nous présenterons une formalisation du système afin de modéliser les différents composants, états, actions et mécanismes d'orchestration qui structurent le fonctionnement de l'architecture.

Enfin, nous présenterons également une approche d'évaluation en trois couches permettant d'analyser les performances du système à différents niveaux. Cette méthodologie permet d'obtenir une vision globale du comportement de l'architecture et d'identifier les forces ainsi que les limites de la solution proposée.

3.2 Analyse du contexte

Le développement de systèmes intelligents capables d'exploiter efficacement les données d'entreprise s'inscrit dans un contexte marqué par une croissance massive et continue des volumes de données. Avec la transformation numérique des organisations, les données sont désormais

générées à travers une multitude de sources hétérogènes telles que les bases relationnelles, les systèmes NoSQL, les fichiers CSV, les entrepôts de données analytiques ainsi que les plateformes métier spécialisées.

Dans le secteur des télécommunications, cette problématique devient particulièrement critique en raison du volume important de données produites quotidiennement. Les opérateurs télécoms génèrent continuellement des informations liées aux recharges, aux ventes, aux souscriptions, aux revenus clients et aux activités réseau, réparties sur l'ensemble des régions du pays. Cette richesse informationnelle représente une source stratégique pour la prise de décision, l'analyse des performances et l'optimisation des services.

Cependant, malgré cette abondance de données, leur exploitation reste difficile pour les utilisateurs non techniques. Les systèmes analytiques traditionnels reposent principalement sur des tableaux de bord statiques, des indicateurs prédéfinis et des outils nécessitant une maîtrise des langages de requêtes comme SQL ou Cypher. L'utilisateur doit généralement connaître la structure exacte des bases de données, les relations entre les tables ainsi que les mécanismes de manipulation des données afin d'obtenir les informations recherchées. Cette dépendance technique limite fortement l'accessibilité de l'analyse de données au sein des organisations.

Parallèlement, les avancées récentes dans le domaine de l'intelligence artificielle, notamment les Large Language Models (LLMs) et les systèmes Retrieval-Augmented Generation (RAG), ont profondément transformé les capacités d'interaction homme-machine. Ces modèles permettent désormais de comprendre des requêtes complexes formulées en langage naturel et d'automatiser une partie du raisonnement analytique. Ils ouvrent ainsi la voie à des systèmes conversationnels capables d'assister les utilisateurs dans l'exploration et l'analyse des données sans nécessiter d'expertise technique avancée.

Néanmoins, les approches traditionnelles basées sur des architectures RAG classique présentent plusieurs limitations lorsqu'elles sont appliquées à des environnements de données complexes et multi-sources.

Dans ce contexte, les architectures Agentic RAG apparaissent comme une solution particulièrement adaptée. En répartissant les responsabilités entre plusieurs agents spécialisés, il devient possible d'améliorer la modularité, la scalabilité et la robustesse globale du système.

La solution proposée s'inscrit précisément dans cette dynamique. Elle vise à concevoir un système analytique intelligent capable d'interroger automatiquement des sources de données hétérogènes à partir de requêtes en langage naturel, puis de générer des réponses analytiques contextualisées accompagnées de visualisations adaptées.

3.3 Étude de l'existant

Avant de détailler la solution proposée, il convient d'examiner les approches existantes permettant d'interroger des données en langage naturel, afin de situer précisément la contribution de ce travail par rapport à l'état actuel des pratiques industrielles et académiques.

3.3.1 Solutions commerciales de BI conversationnelle

Les principaux éditeurs de Business Intelligence ont récemment intégré des couches d'intelligence artificielle générative à leurs plateformes. Microsoft Power BI Copilot permet d'interroger les jeux de données en langage naturel, mais reste dépendant d'un modèle sémantique préalablement construit et nécessite une licence Premium ou Fabric pour ses fonctionnalités avancées. ThoughtSpot, via son agent Spotter (anciennement Sage), adopte une approche de recherche conversationnelle traduisant les requêtes en SQL grâce à un moteur NL2SQL, mais exige une modélisation sémantique préalable dans un langage propriétaire, ce qui représente un investissement de plusieurs semaines avant toute mise en production. Tableau Pulse/Agent et Qlik Answers suivent une logique similaire, en s'appuyant sur des métriques et un modèle sémantique déjà curés par les équipes BI.

Ces solutions partagent une limite structurelle commune : elles opèrent efficacement sur des données déjà modélisées et gouvernées, mais peinent à traiter des questions exploratoires portant sur des sources non cartographiées au préalable. Aucune d'entre elles ne propose nativement une architecture multi-agents capable de router dynamiquement une requête vers des sources hétérogènes (relationnelles, NoSQL, documentaires) sans configuration manuelle extensive.

3.3.2 Approches académiques de type Text-to-SQL / NL2SQL

Un second courant de travaux, davantage académique, se concentre sur la traduction directe du langage naturel en requêtes structurées (Text-to-SQL). Ces approches, bien qu'efficaces sur des bancs d'essai standardisés (Spider, BIRD, etc.), reposent généralement sur un schéma unique et statique, et ne considèrent pas la problématique de sources multiples et hétérogènes typique des environnements d'entreprise réels. Elles ne traitent pas non plus les étapes en aval : agrégation des résultats, génération d'analyses textuelles contextualisées et production de visualisations, qui sont pourtant essentielles à l'expérience utilisateur d'un outil de BI conversationnelle.

3.3.3 Travaux de recherche sur l'Agentic RAG

Sur le plan théorique, la littérature récente sur le RAG Agentique et les architectures modulaires de type "lego" pose les fondements conceptuels d'une décomposition multi-agents du pipeline RAG. Ces travaux restent cependant majoritairement centrés sur des cas d'usage documentaires ou de question-réponse générique, et n'abordent que peu l'application spécifique à l'interrogation de bases de données d'entreprise hétérogènes couplée à la génération de visualisations analytiques.

Ce comparatif met en évidence un manque commun aux solutions existantes : l'absence d'une architecture à la fois modulaire, multi-sources et multi-agents, capable de combiner récupération sémantique des schémas, génération de requêtes validées et synthèse analytique contextualisée, dans un environnement de données d'entreprise réel et hétérogène tel que celui du secteur des télécommunications. C'est précisément ce positionnement que ce travail cherche à occuper, en s'appuyant sur l'architecture Agentic RAG décrite dans le chapitre 2 et détaillée dans la suite de ce chapitre.

3.4 Présentation du système proposé

Le système proposé est un chatbot analytique conçu pour intégrer des sources de données hétérogènes à travers une architecture modulaire et scalable.

Dans les approches conventionnelles, un seul agent est responsable de la compréhension de la requête, de la récupération des données, de la génération des requêtes et de la synthèse de la réponse. Cette architecture centralisée entraîne souvent des problèmes de scalabilité, une utilisation inefficace des ressources, une consommation excessive de tokens ainsi qu'une faible adaptabilité face à des modèles de données variés tels que les bases relationnelles, documentaires, graphe ou les moteurs de recherche.

Afin de surmonter ces limitations, le système proposé adopte une architecture agentique décentralisée dans laquelle plusieurs agents spécialisés collaborent afin de traiter efficacement les requêtes analytiques. Chaque agent est conçu pour gérer une tâche spécifique ou un type particulier de base de données.

En complément, le système intègre un mécanisme sémantique de Retrieval-Augmented Generation (RAG) basé sur des embeddings vectoriels et un index FAISS afin de récupérer les schémas et métadonnées les plus pertinents avant la génération des requêtes. Ce processus de récupération contextuelle améliore la pertinence et la précision des requêtes générées en fournissant au LLM un contexte sémantique structuré.

Une fois les données récupérées, les résultats sont agrégés puis transmis au composant génératif chargé de produire des analyses textuelles cohérentes ainsi que des représentations visuelles adaptées. Grâce à son architecture modulaire, le système proposé améliore la scalabilité, augmente la précision des requêtes, réduit les surcharges computationnelles inutiles et permet un traitement analytique efficace dans des environnements de données complexes et multi-sources.

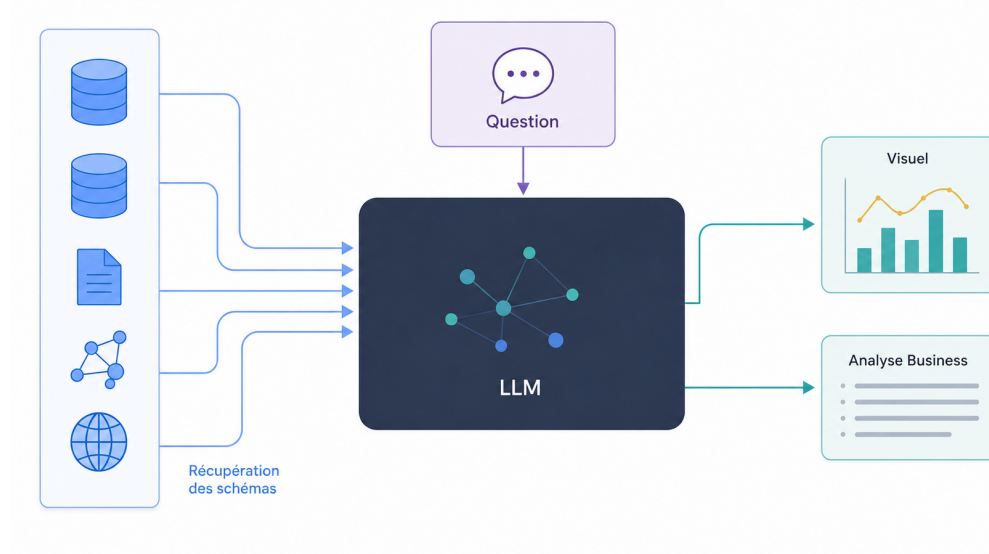


FIG. 3.1 : Présentation du système proposé

3.5 Conception de l'architecture du système

Le système proposé introduit une architecture hybride et modulaire fondée sur une collaboration dynamique entre plusieurs agents spécialisés capables d'interroger simultanément des bases de données hétérogènes. Conçu pour répondre aux besoins analytiques complexes de Djezzy, le système unifie au sein d'un même pipeline des sources relationnelles, documentaires, orientées graphes et textuelles afin de fournir des réponses analytiques cohérentes à partir de requêtes formulées en langage naturel. L'architecture repose sur plusieurs composantes interconnectées opérant de manière séquentielle et coopérative : le module de récupération sémantique des schémas (Schema RAG), l'agent orchestrateur, les agents spécialisés de génération de requêtes, l'environnement d'exécution multi-bases, ainsi que le module de fusion et de génération des réponses. Cette organisation permet de transformer progressivement une question utilisateur en une réponse contextualisée fondée sur des données réelles.

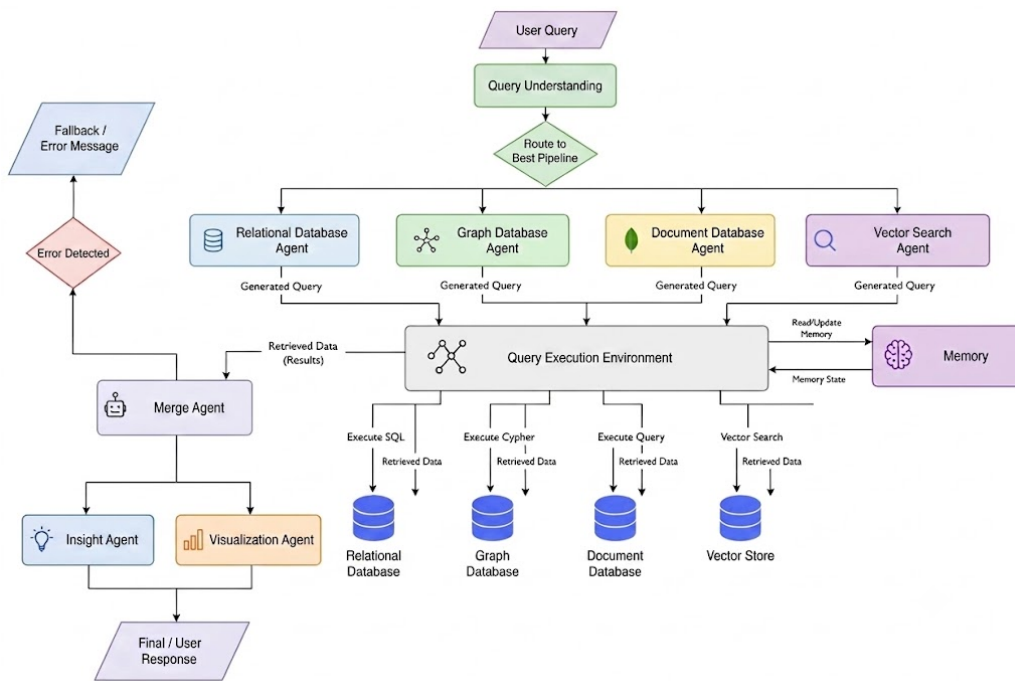


FIG. 3.2 : Vue d'ensemble de l'architecture

3.5.1 Formalisation du système proposé

Afin de décrire formellement l'architecture proposée, cette section présente d'abord les différents composants du système ainsi que leurs interactions. Elle introduit ensuite une classification taxonomique permettant de situer le système par rapport aux principales approches de la littérature.

Formalisation des composants

Le fonctionnement et la structure du système multi-agents proposé peuvent être modélisés à travers les concepts formels suivants :

Agents : L'ensemble des composants du système, noté I , est constitué de plusieurs agents spécialisés assurant des rôles distincts dans le traitement des requêtes utilisateur. Parmi eux figurent : l'agent de détection d'intention, l'orchestrateur principal, les agents spécialisés pour SQLite, MongoDB, Neo4j et ElasticSearch, ainsi que les agents de fusion, d'analyse et de visualisation. Cette organisation modulaire permet une séparation claire des responsabilités et facilite l'extensibilité du système.

$$I = \{i_{\text{Intent}}, i_{\text{Orch}}, i_{\text{SQL}}, i_{\text{Mongo}}, i_{\text{Neo4j}}, i_{\text{ES}}, i_{\text{Merger}}, i_{\text{Viz}}, i_{\text{Insight}}\} \quad \text{avec} \quad |I| = 9$$

$$I = I_{\text{routage}} \cup I_{\text{planif}} \cup I_{\text{génération}} \cup I_{\text{synthèse}}$$

États : Les états du système, notés S , regroupent l'ensemble des ressources et contextes manipulés durant l'exécution. Ils incluent les différentes bases de données hétérogènes telles que SQLite, MongoDB, Neo4j et ElasticSearch, l'index vectoriel FAISS utilisé pour la récupération sémantique des schémas, ainsi que l'historique conversationnel conservant le contexte des interactions précédentes.

Un état global $s \in S$ est un produit cartésien de quatre sous-espaces :

$$S = S_{\text{DB}} \times S_{\text{FAISS}} \times S_{\text{Conv}} \times S_{\text{Exec}}$$

1. Bases de données hétérogènes

Le sous-espace des données est défini par :

$$S_{\text{DB}} = S_{\text{SQLite}} \times S_{\text{Mongo}} \times S_{\text{Neo4j}} \times S_{\text{ES}}$$

Chaque composante représente l'état courant d'une base de données donnée, comprenant à la fois son schéma et son contenu. Ces états sont considérés comme relativement stables durant une session, les agents se limitant à les consulter sans les modifier.

2. Index vectoriel sémantique

Le sous-espace associé à la mémoire vectorielle contient :

- les chunks parents, qui fournissent une vue globale des tables et collections candidates .
- les chunks enfants, qui décrivent les attributs et colonnes exploitables avec une granularité plus fine .
- le vecteur d'embedding de la requête courante, utilisé pour effectuer la recherche sémantique.

Ce sous-espace représente ainsi l'état du mécanisme de récupération de connaissances basé sur FAISS.

3. Historique conversationnel

Le sous-espace conversationnel regroupe :

- l'ensemble des questions précédemment posées .
- les réponses associées .
- la dernière intention détectée par l'agent de classification des intentions.

Il permet au système de conserver le contexte du dialogue et de gérer les interactions multi-tours.

4. État d'exécution courant

Le sous-espace d'exécution est défini par :

$$S_{\text{Exec}} = \text{Plan}_t \times \text{Query}_t \times \text{Retry}_t \times \text{Result}_t$$

où :

- Plan_t correspond au plan d'exécution généré par l'orchestrateur .
- Query_t représente les requêtes produites ou en cours de correction .
- Retry_t désigne le compteur de tentatives de réécriture, borné à trois essais .
- Result_t contient les résultats obtenus auprès des différentes bases de données.

Cette composante constitue la partie la plus dynamique du système puisqu'elle évolue à chaque étape du raisonnement agentique.

Les sous-espaces S_{DB} et S_{FAISS} représentent essentiellement des états observés par les agents. À l'inverse, S_{Exec} constitue l'état actif du système : c'est principalement cette composante qui est modifiée par la fonction de transition $\Psi(s_{t+1} \mid s_t, a_t)$ lors de l'exécution des actions.

(A) Actions : Les actions possibles du système, notées A , correspondent aux différentes opérations exécutées par les agents tout au long du workflow. Ces actions comprennent la classification de l'intention utilisateur, la récupération des schémas pertinents, la génération des requêtes adaptées au type de base de données, la validation de sécurité des requêtes, leur exécution, la fusion des résultats, la génération de visualisations ainsi que la production de la réponse finale.

(Ω) Observations : Les observations du système, notées Ω , représentent les informations perçues après chaque étape d'exécution. Elles incluent les résultats des requêtes exécutées, les éventuelles erreurs d'exécution et les fragments de schémas récupérés via le mécanisme RAG.

(O) Fonction d'observation : La fonction d'observation, notée O , est principalement assurée par le module Schema RAG basé sur FAISS. Ce mécanisme effectue une recherche sémantique afin de filtrer les schémas les plus pertinents selon la requête utilisateur. Ainsi, chaque agent accède uniquement aux informations nécessaires à sa tâche, ce qui améliore la précision des requêtes générées tout en réduisant le bruit informationnel.

(II) Politiques : Les politiques du système, notées Π , reposent sur le modèle de langage Llama. Chaque agent dispose d'un prompt spécialisé adapté à sa fonction spécifique dans l'architecture. Cette spécialisation des prompts permet d'orienter le comportement des agents selon leur rôle et d'assurer une meilleure cohérence des traitements.

(M) Mémoire : Le mécanisme de mémoire, noté M , est une mémoire conversationnelle à court terme conserve les derniers échanges utilisateurs à travers une structure `conversation_history` limitée à dix tours afin de maintenir la continuité contextuelle.

(Ψ) **Fonction de transition** : La fonction de transition, notée Ψ , est implémentée à travers le graphe LangGraph qui orchestre l'ensemble du workflow. Chaque nœud du graphe modifie dynamiquement le **GraphState** en fonction des résultats intermédiaires obtenus, tandis que les routeurs conditionnels déterminent le prochain nœud à exécuter selon l'état courant et l'intention détectée. Cette approche permet une orchestration flexible et adaptative du système multi-agents.

Taxonomie du système

Afin de situer l'architecture proposée au sein de la littérature actuelle sur les systèmes basés sur les grands modèles de langage, le tableau 3.1 classe le système selon ses dimensions taxonomiques fondamentales.

TAB. 3.1 : Classification du système selon les dimensions taxonomiques

Dimension taxonomique	Taxonomie liée (Concepts clés)
Architecture	Planner–Executor
Stratégie de Récupération	Récupération itérative avec mécanisme réflexif partiel
Paradigme de Raisonnement	ReAct (<i>Reasoning and Acting</i>)
Gestion de la Mémoire	Mémoire épisodique : Historique de conversation (cohérence locale sur 10 tours)

Justification de la Taxonomie

Après avoir identifié les principales dimensions taxonomiques du système, il est nécessaire de justifier ce classement.

Architecture : Planner–Executor

Le système repose sur une séparation explicite et structurelle entre la planification et l'exécution. L'agent Orchestrateur incarne le rôle de Planner : il raisonne sur la nature de la question, les bases disponibles et le contexte schématique pour élaborer un plan d'exécution complet. Ce plan est ensuite délégué aux agents spécialisés (Executors), chacun chargé de produire et d'exécuter une requête dans le langage natif de sa base cible.

Cette dissociation est caractéristique du paradigme Planner–Executor, où aucun agent exécuter ne décide de sa propre activation : c'est le plan produit par l'Orchestrateur qui active les agents correspondants.

La granularité de cette séparation est renforcée par la présence d'agents terminaux distincts (Viz, Insight, Merger), chacun exécutant une étape atomique du plan global, sans connaissance

de l'ensemble de la chaîne.

Stratégie de Récupération : Récupération itérative avec mécanisme réflexif partiel

La récupération ne s'effectue pas en une seule passe statique. Le système engage d'abord un Schema RAG par recherche de similarité cosinus dans un index FAISS hiérarchique, retournant à la fois des chunks parents (vue d'ensemble des tables) et des chunks enfants (colonnes exactes). Ce contexte ciblé conditionne l'ensemble des décisions en aval. Le caractère itératif avec réflexivité partielle se manifeste dans le mécanisme de query rewriting guidé par l'erreur : lorsqu'une requête générée est invalide ou produit une erreur d'exécution, le message d'erreur est réinjecté comme signal correctif dans une nouvelle invocation du même agent générateur. Ce cycle peut se répéter jusqu'à trois fois par agent. Il s'agit d'une récupération itérative (chaque tentative s'appuie sur le résultat de la précédente) mais la réflexivité reste partielle, car elle est bornée (3 tentatives max) et locale à l'agent concerné, sans raisonnement global sur l'ensemble du plan.

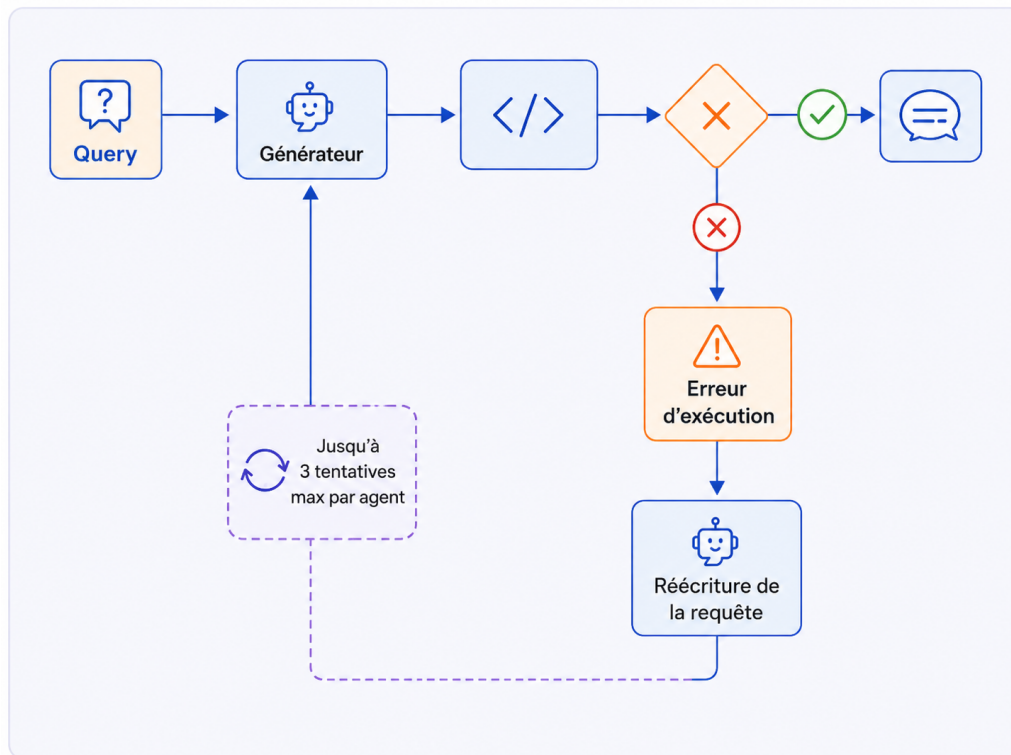


FIG. 3.3 : Schéma du processus de récupération itérative

Paradigme de Raisonnement : ReAct

Le paradigme ReAct se définit par l'alternance de cycles Thought $\rightarrow$ Action $\rightarrow$ Observation. Ce pattern est structurellement présent à deux niveaux du système. Au niveau de l'Orchestrateur, celui-ci raisonne explicitement sur la nature de la question et les bases disponibles (Thought), puis agit en produisant un plan d'exécution (Action). Au niveau des agents générateurs, le mécanisme réflexif instancie directement ce cycle : l'agent raisonne sur sa requête originale et le message d'erreur reçu (Thought), produit une requête corrigée (Action), et reçoit en retour

le résultat du validateur ou de la base (Observation). Par ailleurs, la classification d'intention (Intent agent) constitue une étape de raisonnement préalable à toute action, caractéristique des architectures ReAct où l'agent évalue la situation avant de décider d'agir. La structure LangGraph, avec ses nœuds de décision et d'action spécialisés, formalise ce paradigme au niveau du graphe d'exécution lui-même.

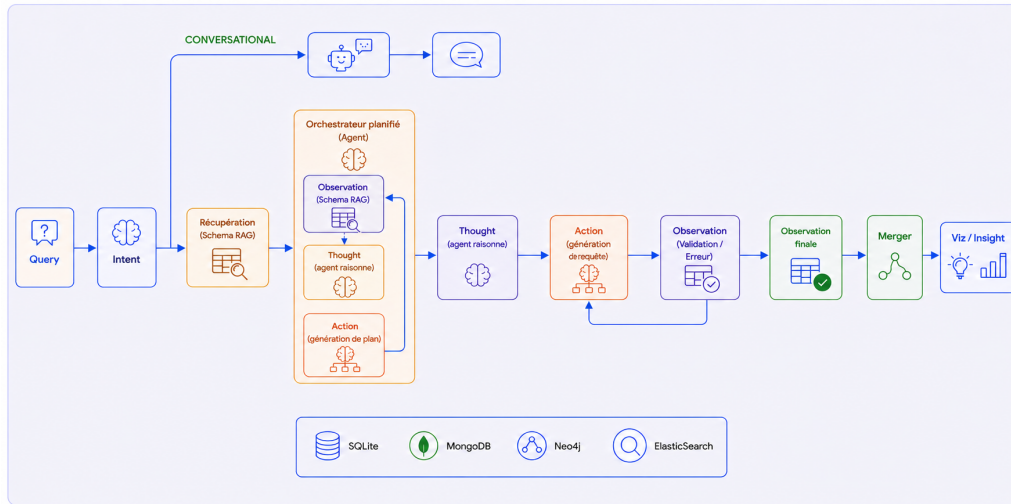


FIG. 3.4 : Paradigme de raisonnement : ReAct

Gestion de la Mémoire : Mémoire épisodique

La mémoire mobilisée par le système est de nature épisodique : elle enregistre la séquence des échanges passés entre l'utilisateur et le système au sein d'une même session, sans persistance inter-sessions ni apprentissage à long terme. Cette mémoire alimente la cohérence contextuelle du pipeline (notamment pour l'agent Intent et l'agent Insight) en permettant de situer une nouvelle question dans la continuité des échanges précédents. La fenêtre de 10 tours définit l'horizon épisodique exploitable : seuls les dix derniers échanges sont retenus comme contexte actif, ce qui constitue une cohérence locale bornée. Ce choix architectural est justifié par le compromis entre la richesse contextuelle nécessaire à la pertinence des réponses et le coût en tokens d'un historique illimité.

3.5.2 Modules principaux de l'architecture

Cette section présente les principaux modules de l'architecture proposée. Le système est basé sur une approche modulaire de type RAG (Retrieval-Augmented Generation), où chaque composant joue un rôle précis dans le traitement d'une requête utilisateur.

3.5.2.1 Classification d'intention

Avant toute interrogation des bases de données, l'agent d'intention analyse la question utilisateur et la catégorise selon deux classes : une requête analytique (**DATA_QUERY**) nécessitant l'activation du pipeline complet, ou une interaction conversationnelle (**CONVERSATIONAL**).

traitée directement sans solliciter les modules de génération de requêtes. Cette bifurcation précoce conditionne l'ensemble du chemin d'exécution qui suit.

$$\mathcal{I}_{intent} = \mathcal{F}_{intent}(Q_{user}, H) \quad \text{avec} \quad \mathcal{I}_{intent} \in \{\text{DATA_QUERY}, \text{CONVERSATIONAL}\} \quad (3.1)$$

Où :

- $\mathcal{F}_{intent}$ représente la fonction de classification basée sur le LLM,
- Q_{user} est la question soumise par l'utilisateur,
- H est l'historique conversationnel fourni en contexte.

3.5.2.2 Récupération sémantique des schémas

Avant toute génération de requête, le système doit déterminer quelles structures de données sont pertinentes pour répondre à la question utilisateur. Cette étape est assurée par le module Schema RAG, chargé d'effectuer une récupération sémantique des schémas des différentes bases de données.

L'ensemble des schémas, descriptions de tables, collections, relations et métadonnées est encodé sous forme vectorielle, puis indexé dans une structure FAISS unifiée. Lorsqu'une requête utilisateur est soumise, le système génère son embedding puis effectue une recherche par similarité cosinus afin d'identifier les fragments de schéma les plus pertinents.

Le processus peut être formalisé comme suit :

$$S_{context} = \text{FAISS-Search}(E(Q_{user}), \mathcal{I}_{schema}, k) \quad (3.2)$$

Où :

- E représente la fonction d'encodage vectoriel,
- $\mathcal{I}_{schema}$ l'index FAISS contenant les schémas,
- k le nombre de fragments retournés.

Cette approche permet de limiter le contexte transmis aux modèles de langage uniquement aux éléments structurels nécessaires, réduisant ainsi la surcharge contextuelle et améliorant la précision des requêtes générées.

3.5.2.3 Agent Orchestrateur

L'agent orchestrateur constitue le noyau décisionnel du système. Il reçoit simultanément la question utilisateur et le contexte sémantique retourné par le module Schema RAG afin de déterminer la stratégie d'exécution optimale.

Son rôle consiste à identifier quelles bases de données doivent être interrogées, dans quel ordre, et selon quelle logique les résultats devront être combinés. L'orchestrateur distingue notamment les requêtes simples ciblant une seule base des requêtes complexes nécessitant une orchestration multi-sources.

Le plan d'exécution généré peut être représenté par :

$$\mathcal{P} = \pi_{orch}(Q_{user}, S_{context}, \mathcal{DB}_{available}) \quad (3.3)$$

Où :

- $\mathcal{P}$ contient la stratégie d'exécution, les bases sélectionnées, la logique de fusion ainsi que les éventuelles clés de jointure,
- π_{orch} représente la politique ou fonction d'orchestration,
- Q_{user} est la requête initiale de l'utilisateur,
- $S_{context}$ est le contexte de schéma récupéré par le RAG,
- $\mathcal{DB}_{available}$ désigne l'ensemble des bases de données disponibles dans le système.

Cette composante permet au système de fonctionner comme un moteur analytique distribué capable d'adapter dynamiquement son comportement selon la nature de la requête utilisateur.

3.5.2.4 Agents de Génération de Requêtes

Afin de garantir une génération de requêtes adaptée à chaque environnement de données, le système repose sur plusieurs agents spécialisés, chacun étant dédié à un type de base spécifique.

Chaque agent traduit la requête utilisateur en une requête exécutable dans le langage natif de la base cible en s'appuyant sur le contexte schématique récupéré précédemment. Le processus de génération peut être exprimé par :

$$Q_{generated} = f_{agent}(Q_{user}, S_{context}, e_{prev}) \quad (3.4)$$

Où :

- $Q_{generated}$ représente la requête finale générée pour la base de données,
- f_{agent} correspond à la fonction ou politique de l'agent spécialisé,
- Q_{user} est la question initiale posée par l'utilisateur,
- $S_{context}$ est le contexte de schéma filtré et fourni par le module Schema RAG,
- e_{prev} représente un éventuel message d'erreur provenant d'une tentative précédente, permettant au système d'effectuer des corrections itératives.

Cette spécialisation améliore considérablement la précision des requêtes et permet au système de s'adapter naturellement à des environnements de données hétérogènes.

3.5.2.5 Environnement d'Exécution et Validation

Avant leur exécution, les requêtes générées traversent une couche de validation destinée à détecter les instructions interdites, vérifier leur cohérence syntaxique et prévenir les opérations dangereuses.

Les requêtes validées sont ensuite transmises à un environnement d'exécution capable de sélectionner dynamiquement le driver correspondant au type de base interrogée.

En cas d'échec, le message d'erreur est réinjecté dans le workflow afin de permettre à l'agent générateur de reformuler automatiquement la requête. Cette boucle réflexive améliore la robustesse globale du système et réduit les échecs d'exécution. Le nombre de tentatives est limité afin d'éviter les boucles infinies et de préserver les performances du système.

3.5.2.6 Module de Fusion et Génération de Réponse

Lorsque plusieurs bases de données sont interrogées simultanément, les résultats obtenus doivent être consolidés afin de produire une réponse unifiée. Cette tâche est assurée par l'agent de fusion (*Merger Agent*), qui applique la stratégie définie par l'orchestrateur.

Plusieurs modes de fusion sont supportés :

- La jointure sur clé commune
- L'union par concaténation
- L'enrichissement contextuel
- L'exécution séquentielle où le résultat d'une première requête alimente les suivantes.

L'agent de visualisation (*Viz Agent*) détermine automatiquement la représentation graphique la plus pertinente selon la structure des données et la sémantique de la requête. Le système peut ainsi produire des graphiques de type barres, courbes, secteurs, nuages de points, aires ou tableaux.

Parallèlement, l'agent d'analyse (*Insight Agent*) génère une interprétation analytique des résultats tout en restant strictement ancré dans les données récupérées afin de limiter les hallucinations du modèle génératif.

Grâce à cette organisation modulaire et spécialisée, l'architecture proposée demeure flexible, scalable et robuste face à des environnements de données complexes et diversifiés. Elle constitue ainsi une solution adaptée aux problématiques de gestion de données hétérogènes dans un contexte Retrieval-Augmented Generation (RAG).

3.6 Pipeline d'exécution du système

Le fonctionnement du système peut être résumé à travers les étapes suivantes.

1. Réception de la question.

L'utilisateur soumet une question en langage naturel. Cette question constitue le point d'entrée du pipeline et déclenche l'ensemble du processus de traitement.

2. Classification de l'intention.

L'agent *Intent* analyse la question afin de déterminer sa nature.

- DATA QUERY
- CONVERSATIONAL

Cette étape permet d'éviter l'exécution du pipeline analytique lorsque la réponse peut être produite directement.

3. Récupération du schéma pertinent.

Pour une requête analytique, le module *Schema RAG* encode la question puis effectue une recherche par similarité dans l'index FAISS.

La recherche retourne les informations les plus pertinentes.

- Les chunks parents décrivant les tables candidates.
- Les chunks enfants contenant les colonnes associées.
- Les types de bases de données les plus adaptés à la requête.

Cette étape permet de fournir au système uniquement les éléments de schéma nécessaires au traitement de la question.

4. Planification de l'exécution.

Le contexte récupéré est transmis à l'agent *Orchestrateur*, qui constitue le cœur décisionnel du système.

À partir de la question et des schémas identifiés, il détermine.

- Les bases de données à interroger.
- L'ordre d'exécution.
- La stratégie de fusion des résultats.

5. Génération des requêtes.

L'Orchestrateur active ensuite les agents spécialisés correspondant aux bases sélectionnées.

- **SQLite** pour la génération de requêtes SQL validées avec `sqlglot`.
- **MongoDB** pour la génération de pipelines d'agrégation.
- **Neo4j** pour la génération de requêtes Cypher.
- **ElasticSearch** pour la génération de requêtes DSL JSON.

Chaque agent reçoit la question utilisateur ainsi que le schéma filtré et produit une requête adaptée à sa base de données. La génération est guidée par des prompts spécialisés garantissant le respect de la structure des schémas, des conventions syntaxiques et des règles métier.

6. Validation et correction automatique.

Chaque requête générée est vérifiée avant son exécution.

Si une erreur est détectée lors de la validation ou de l'exécution, le système récupère le message d'erreur et le transmet à l'agent concerné afin qu'il produise une nouvelle version corrigée de la requête.

Ce mécanisme de *query rewriting* permet notamment de corriger.

- Les erreurs de syntaxe.
- Les colonnes inexistantes.
- Les opérateurs non supportés.

7. Exécution des requêtes.

Les requêtes validées sont exécutées à l'aide du driver correspondant à chaque type de base de données.

Les résultats obtenus sont ensuite collectés et préparés pour l'étape suivante.

8. Fusion des résultats.

Lorsque plusieurs bases ont été interrogées, l'agent *Merger* combine les résultats selon la stratégie définie par l'Orchestrateur.

9. Génération des visualisations et des analyses.

Deux agents interviennent en fin de pipeline.

- **Viz**, qui sélectionne automatiquement la visualisation la plus pertinente.
- **Insight**, qui génère un commentaire analytique à partir des données obtenues.

L'agent *Insight* construit d'abord une synthèse factuelle basée uniquement sur les données, puis utilise le LLM pour produire une interprétation des résultats. Cette séparation permet de réduire les hallucinations tout en conservant la capacité d'analyse du modèle.

10. Production de la réponse finale.

La réponse retournée à l'utilisateur comprend.

- L'analyse textuelle.
- Les visualisations générées.
- Les requêtes exécutées.
- Les sources de données utilisées.

Cette approche améliore la transparence du système en permettant à l'utilisateur de comprendre l'origine des résultats et les différentes étapes du raisonnement.

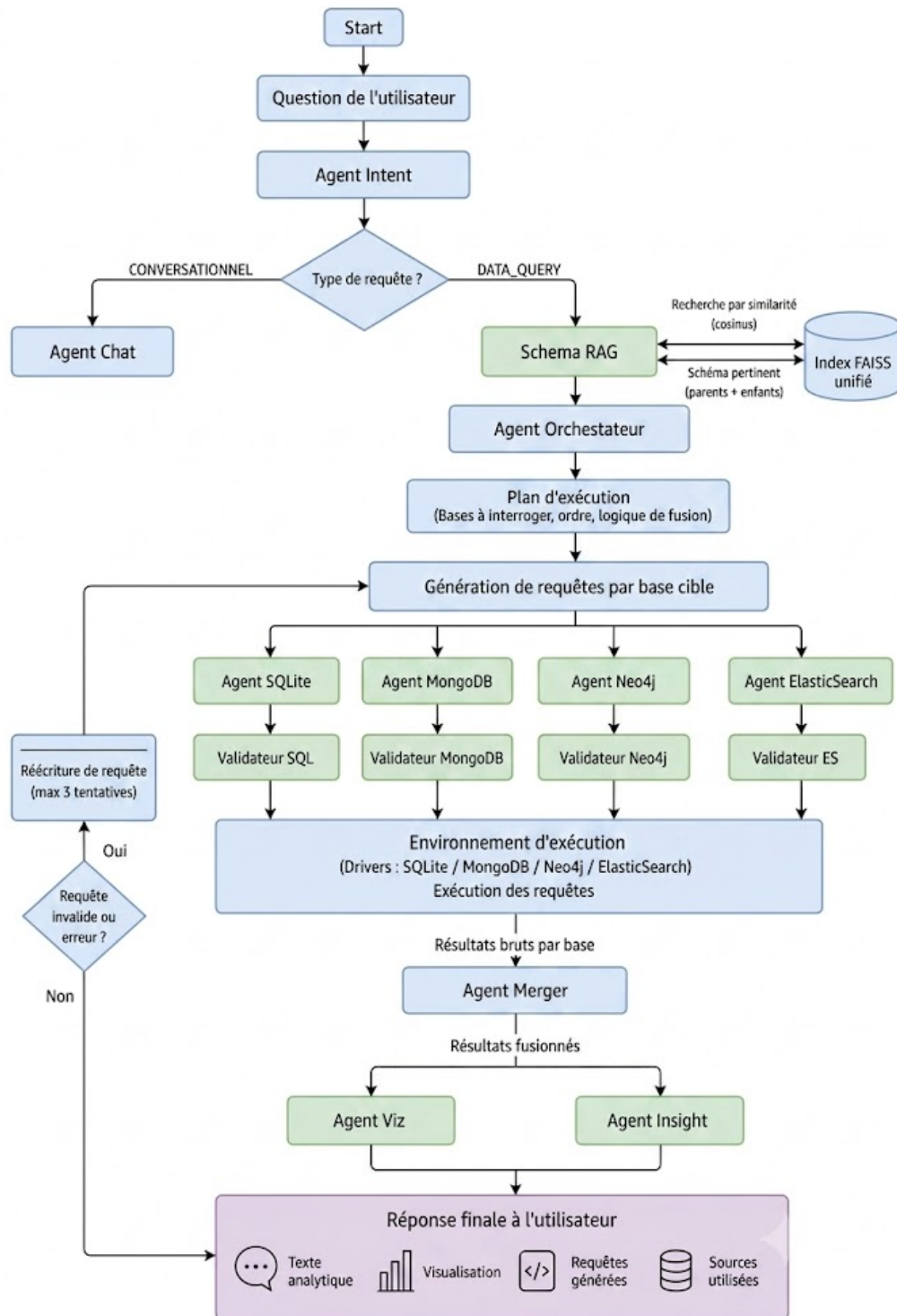


FIG. 3.5 : Pipeline d'exécution du système

3.7 Implémentation de la Solution

Cette section présente les choix de conception et les éléments techniques ayant permis l'implémentation du système. Elle décrit d'abord les algorithmes clés utilisés pour le traitement des requêtes, l'exécution et la fusion des résultats. puis structure le pipeline global, puis détaille les principaux choix architecturaux effectués.

Enfin, elle aborde les techniques d'ingénierie des prompts mises en place pour garantir la fiabilité, la précision et la cohérence des sorties générées par les différents agents du système.

3.7.1 Méthodologie de développement

3.7.1.1 Préparation des données

Contrairement aux approches classiques nécessitant une phase de collecte, de nettoyage et de chargement des données (*ETL*), notre système s'appuie sur des données déjà traitées et stockées dans leurs bases de données respectives. Les données sont donc considérées comme préalablement préparées et exploitables au moment de leur intégration.

La préparation se limite donc à l'établissement des connexions vers ces sources. Pour chaque base de données, un driver natif est utilisé avec les paramètres de connexion correspondants (URL, port, identifiants) : `sqlite3` pour SQLite, `pymongo` pour MongoDB, `neo4j` pour Neo4j, et `elasticsearch-py` pour Elasticsearch. Ces connexions sont initialisées au démarrage du système.

3.7.1.2 Indexation des schémas

La première étape du système consiste à construire un registre de schémas regroupant les métadonnées de toutes les sources de données connectées. Pour chaque base de données, le schéma est découpé en plusieurs fragments textuels : un fragment pour chaque table ou collection, puis un autre pour chaque colonne ou champ. Chaque fragment contient des informations comme le type de base, le nom de la table, le type de données et quelques valeurs d'exemple.

Ces fragments sont ensuite transformés en *embeddings* avec le modèle `all-mpnet-base-v2`, puis stockés dans un index FAISS de type HNSW utilisant la similarité cosinus. Cette étape permet d'effectuer une recherche sémantique unifiée sur plusieurs bases de données hétérogènes.

Chaque fragment garde une métadonnée `db_type` afin de conserver l'origine de la base pendant tout le pipeline.

Algorithme 1 : Indexation des schémas**Entrée :** Sources de données $S = \{\text{SQLite, MongoDB, Neo4j, Elasticsearch}\}$ **Sortie :** Index_FAISS, Registre_ALL_SCHEMAS**foreach** *source* $s \in S$ **do**

- | SchémaChunks $\leftarrow$ décomposer_schéma(s) // un chunk par table/collection + un par colonne/champ

- | Annoter chaque chunk avec {db_type, table, texte, valeurs_exemples}

Embeddings $\leftarrow$ SentenceTransformer.encoder(SchémaChunks)Index_FAISS $\leftarrow$ HNSW.construire(Embeddings)**3.7.1.3 Agent de classification d'intention**

Chaque requête utilisateur passe d'abord par un agent de classification d'intention avant toute interaction avec les bases de données. L'agent reçoit le message de l'utilisateur ainsi que l'historique de la conversation, puis décide si la requête nécessite réellement un accès aux données ou s'il s'agit simplement d'un échange conversationnel. Deux catégories sont possibles : **DATA_QUERY** pour les questions liées aux données, et **CONVERSATIONAL** pour les discussions classiques. Cette étape permet de choisir rapidement le bon pipeline à exécuter.

Algorithme 2 : Classification d'intention**Entrée :** Message utilisateur Q , Historique H **Sortie :** IntentionPrompt $\leftarrow$ construire_prompt(Q, H)Résultat $\leftarrow$ LLM.inférer(Prompt)**if** *Résultat valide* **then**

- | Intention $\leftarrow$ extraire_json(Résultat).intent // "DATA_QUERY" | "CONVERSATIONAL"

else

- | Intention $\leftarrow$ heuristique_lexicale(Q) // repli par mots-clés

3.7.1.4 Récupération sémantique des schémas

Quand une requête est classifiée comme **DATA_QUERY**, le système lance une recherche sémantique dans l'index FAISS afin de retrouver les parties de schéma les plus pertinentes. L'embedding de la question utilisateur est calculé puis comparé à tous les vecteurs présents dans l'index. Les résultats les plus proches sont récupérés, puis filtrés pour éviter les doublons entre tables et types de bases.

Le contexte obtenu est ensuite envoyé à l'agent orchestrateur avec la liste des bases de données concernées.

Algorithme 3 : Récupération sémantique des schémas**Entrée :** Message utilisateur Q , Index_FAISS, SchémaChunks, $k = \text{TOP_K}$ **Sortie :** ContexteSchéma, TypesBases_TrouvésEmbedding_Q $\leftarrow$ SentenceTransformer.encoder(Q)Indices $\leftarrow$ FAISS.rechercher(Embedding_Q, k)**foreach** *indice* $i \in \text{Indices}$ **do** chunk $\leftarrow$ SchémaChunks[i] **if** *chunk non déjà vu (table, db_type)* **then**

AjouterÀRésultats(chunk)

AjouterÀ(TypesBases_Trouvés, chunk.db_type)

ContexteSchéma $\leftarrow$ concaténer(Résultats)**3.7.1.5 Agent orchestrateur**

L'agent orchestrateur est le composant principal du système. Il reçoit la requête utilisateur, le contexte des schémas récupérés et les types de bases détectés, puis construit un plan d'exécution au format JSON.

Ce plan indique quelle stratégie utiliser : **single** lorsqu'une seule base suffit, ou **multi** lorsqu'il faut combiner plusieurs sources. L'agent précise également quelles bases doivent être interrogées, comment les résultats doivent être fusionnés et, si nécessaire, quelle clé utiliser pour la jointure.

Algorithme 4 : Orchestration et planification**Entrée :** Message Q , ContexteSchéma, TypesBases_Trouvés**Sortie :** Plan_Orchestration, BaseActivesPrompt $\leftarrow$ construire_prompt_orchestrateur(Q , ContexteSchéma, TypesBases_Trouvés)PlanJSON $\leftarrow$ LLM.inférer(Prompt)Plan $\leftarrow$ extraire_json(PlanJSON)Stratégie $\leftarrow$ Plan.strategy

// "single" | "multi"

BaseActives $\leftarrow$ [db.db_type pour db dans Plan.databases]LogiqueFusion $\leftarrow$ Plan.merge_logic

// "union" | "join" | "enrichissement"

CléFusion $\leftarrow$ Plan.merge_key**3.7.1.6 Agents de génération de requêtes**

Le système utilise un agent spécialisé pour chaque type de base de données identifié dans le plan d'orchestration. Chaque agent reçoit la question utilisateur, le contexte de schéma correspondant et, en cas d'erreur précédente, le message d'erreur associé.

Selon la base ciblée, l'agent génère une requête adaptée : SQL pour SQLite, pipeline d'agrégation pour MongoDB, requête Cypher pour Neo4j ou requête DSL pour Elasticsearch. Tous les agents sont configurés pour éviter toute opération destructrice sur les données.

Algorithme 5 : Génération des requêtes**Entrée :** Message Q , ContexteSchéma, BaseActives, Erreurs = {}**Sortie :** RequêtesParBase

```

foreach  $db\_type \in BaseActives$  do
    Agent  $\leftarrow$  sélectionner_agent( $db\_type$ )           // SQLite | MongoDB | Neo4j | ES
    Erreur  $\leftarrow$  Erreurs.get( $db\_type$ , "")
    Requête  $\leftarrow$  Agent.générer( $Q$ , ContexteSchéma, Erreur)
    RequêtesParBase[ $db\_type$ ]  $\leftarrow$  Requête

```

3.7.1.7 Agent de validation

Avant l'exécution, toutes les requêtes passent par un agent de validation. Pour SQLite, le système utilise la bibliothèque `sqlglot` afin de vérifier la syntaxe et détecter les opérations interdites comme DROP, DELETE, INSERT ou UPDATE.

Pour les autres bases, une vérification simple est appliquée afin de s'assurer que la requête n'est pas vide. Si une requête est invalide, elle est renvoyée à l'agent de génération avec le message d'erreur pour correction. Le système autorise jusqu'à trois tentatives avant de retourner une erreur finale.

Algorithme 6 : Validation et gestion des reprises**Entrée :** RequêtesParBase, BaseActives, CompteurRetry**Sortie :** {valide $\rightarrow$ Exécution | retry $\rightarrow$ Génération | échec $\rightarrow$ Erreur}ToutValide $\leftarrow$ Vrai

```

foreach  $db\_type \in BaseActives$  do
    Requête  $\leftarrow$  RequêtesParBase[ $db\_type$ ]
    if  $db\_type == "sqlite"$  then
        Résultat  $\leftarrow$  sqlglot.valider(Requête)
        if Résultat invalide ou non-sécurisé then
            Erreurs[ $db\_type$ ]  $\leftarrow$  Résultat.erreur
            ToutValide  $\leftarrow$  Faux
    else
        if Requête vide then
            Erreurs[ $db\_type$ ]  $\leftarrow$  "requête vide"
            ToutValide  $\leftarrow$  Faux

```

if ToutValide **then**

| Renvoyer Exécution

else| **if** CompteurRetry < 3 **then**

| | Renvoyer Génération (retry)

| **else**

| | Renvoyer Erreur

3.7.1.8 Agent d'exécution

Une fois validées, les requêtes sont exécutées à l'aide des connecteurs adaptés : `sqlite3` pour SQLite, `pymongo` pour MongoDB, le driver `neo4j` pour Neo4j et le client Python `elasticsearch` pour Elasticsearch.

Chaque résultat est ensuite converti dans un format commun basé sur une liste de dictionnaires afin de simplifier les traitements suivants. Les erreurs d'exécution sont gérées séparément pour chaque base afin qu'un problème sur une source n'interrompe pas les autres.

Algorithme 7 : Exécution des requêtes

Entrée : RequêtesParBase, BaseActives

Sortie : RésultatsParBase, SourcesUtilisées

```

foreach db_type ∈ BaseActives do
    Connecteur ← obtenir_connecteur(db_type)
    Résultat ← Connecteur.exécuter(RequêtesParBase[db_type])
    if Résultat.succès then
        RésultatsParBase[db_type] ← normaliser(Résultat.données)
        AjouterÀ(SourcesUtilisées, db_type)
    else
        Erreurs[db_type] ← Résultat.erreur
        RésultatsParBase[db_type] ← []

```

3.7.1.9 Agent de fusion

Lorsque plusieurs bases de données sont utilisées, un agent de fusion combine les résultats obtenus. Selon la stratégie définie dans le plan d'orchestration, il peut effectuer une union simple, une jointure sur une clé commune ou un enrichissement d'un résultat par un autre.

Pour les cas plus complexes, l'agent peut utiliser le LLM afin d'interpréter les relations entre les données récupérées. Si une seule base est utilisée, les résultats sont directement transmis sans modification.

Algorithme 8 : Fusion des résultats

Entrée : RésultatsParBase, Plan_Orchestration, Message *Q*

Sortie : RésultatsFusionnés, TypeRésultat

```

Stratégie ← Plan_Orchestration.strategy
if Stratégie == "single" ou |RésultatsParBase| ≤ 1 then
    RésultatsFusionnés ← RésultatsParBase[première_source]
else
    LogiqueFusion ← Plan_Orchestration.merge_logic
    CléFusion ← Plan_Orchestration.merge_key
    RésultatsFusionnés ←
        AgentFusion.combiner(RésultatsParBase, LogiqueFusion, CléFusion, Q)
TypeRésultat ← classifier_résultat(RésultatsFusionnés) // vide | scalaire | ligne |
tableau

```

3.7.1.10 Agent de visualisation

L'agent de visualisation choisit automatiquement le type de représentation graphique le plus adapté aux résultats obtenus. Il analyse la structure des données, le nombre de colonnes, les types de valeurs ainsi que la question posée par l'utilisateur.

À partir de cela, il génère une configuration contenant le type de graphique à afficher (barres, courbe, nuage de points ou tableau), les axes, le titre et les données nécessaires au rendu côté frontend.

Algorithme 9 : Génération de la visualisation

Entrée : RésultatsFusionnés, TypeRésultat, Message Q

Sortie : Config_Visualisation

Prompt $\leftarrow$ construire_prompt_viz(RésultatsFusionnés, TypeRésultat, Q)

ConfigJSON $\leftarrow$ LLM.inférer(Prompt)

Config $\leftarrow$ extraire_json(ConfigJSON)

Config $\leftarrow$ {type_chart, titre, axe_x, axe_y, données}

3.7.1.11 Agent d'insight et de génération de réponse

Dans la dernière étape, l'agent d'insight génère une réponse finale en langage naturel à partir des résultats fusionnés, de la configuration de visualisation, des bases utilisées et de la requête initiale.

La réponse finale contient le texte analytique, la configuration du graphique, les requêtes générées pour chaque base ainsi que les sources utilisées. L'historique de conversation est conservé tout au long du processus afin de maintenir le contexte entre les différentes interactions.

Algorithme 10 : Génération de l'insight et de la réponse finale

Entrée : RésultatsFusionnés, Config_Visualisation, Message Q , SourcesUtilisées, RequêtesParBase

Sortie : RéponseFinal

Prompt $\leftarrow$

construire_prompt_insight(RésultatsFusionnés, Config_Visualisation, Q , SourcesUtilisées)

TexteInsight $\leftarrow$ LLM.inférer(Prompt)

RéponseFinal $\leftarrow$ {

type : "data",

message : TexteInsight,

chart : Config_Visualisation,

queries : RequêtesParBase,

db_sources : SourcesUtilisées,

row_count : |RésultatsFusionnés|

}

3.7.2 Ingénierie des Prompts

L'ingénierie des prompts constitue l'une des décisions techniques les plus déterminantes du système. Chaque agent reçoit un prompt système hautement spécialisé, conçu pour contraindre le modèle à produire des sorties fiables . 7 principes ont guidé la conception de l'ensemble des prompts.

3.7.2.1 Assignation de rôle

Chaque agent reçoit en premier lieu une assignation de rôle explicite qui conditionne le registre et les attentes du modèle. Plutôt qu'un rôle générique, chaque persona est ancré dans le domaine métier précis :

- *"You are an expert SQLite query generator."*
- *"You are a world-class Neo4j Cypher expert working on a telecom fraud detection graph database (Djezzy)."*
- *"You are a senior telecom business analyst working for Djezzy Algeria."*
- *"You are the central query orchestrator for Djezzy's hybrid multi-database analytics platform."*

Cette technique de *role prompting* permet de : réduire l'espace de génération du modèle en l'ancrant dans une fonction précise, ce qui diminue les réponses hors-sujet et améliore la cohérence des sorties.

3.7.2.2 Injection du contexte structuré

Le schéma récupéré par le Schema RAG est ajouté directement au prompt des agents générateurs. Pour certains agents, comme l'Orchestrateur et l'agent Neo4j, des séparateurs visuels sont utilisés pour distinguer clairement les différentes informations :

```
SCHEMA
{schema}
USER QUESTION
"{user_message}"
```

Cette organisation permet au modèle de mieux distinguer le schéma de la question utilisateur et facilite également l'analyse et le débogage des prompts en rendant le contexte transmis plus lisible.

3.7.2.3 Règles métier explicites et contraintes négatives

Chaque prompt intègre un ensemble de règles positives (ce que le modèle doit faire) et de contraintes négatives (ce qu'il ne doit jamais faire), réduisant ainsi les comportements indésirables sans supervision.

Pour l'agent SQLite :

Rules:

- SELECT statements ONLY (never DROP, DELETE, INSERT, UPDATE)
- Use EXACT column names from the schema
- Never use SELECT *
- Add LIMIT 500 unless it's a scalar aggregation
- Use strftime() for date operations in SQLite

Ce niveau de détail traite des ambiguïtés linguistiques que le modèle ne pourrait pas résoudre à partir du seul schéma, réduisant ainsi les erreurs de sens (trier dans le mauvais ordre, omettre un filtre implicite).

3.7.2.4 Exemples few-shot ciblés

Pour les langages de requête moins courants dans les données d'entraînement des LLM, des exemples *few-shot* sont intégrés directement dans le prompt :

Example for "how many operational 4G antennas in Alger":

```
db.antennes.aggregate([
  {"$match": {"technologie": "4G", "wilaya": "Alger", "statut": "Opérationnelle"}},
  {"$count": "total"}
])
```

Example for "average resolution time by wilaya, top 5 slowest":

```
db.incidents_reseau.aggregate([
  {"$match": {"duree_resolution_min": {"$ne": null}}},
  {"$group": {"_id": "$wilaya", "total": {"$avg": "$duree_resolution_min"}}},
  {"$sort": {"total": -1}},
  {"$limit": 5}
])
```

Ces exemples servent de calibrage syntaxique : ils ancrent le modèle dans le format exact attendu et réduisent les erreurs de structure (oubli du \$ dans les opérateurs, confusion entre `find()` et `aggregate()`).

3.7.2.5 Contrôle du format de sortie et parsing défensif

Un principe transversal à tous les agents est l'exigence de format de sortie strict, accompagnée d'une instruction finale non ambiguë :

- *"Respond with ONLY the raw SQL query, no markdown, no explanation."* (SQLite)
- *"Return ONLY the raw Cypher query — No markdown, no code fences, no backtick blocks"* (Neo4j)
- *"RESPOND WITH JSON ONLY — no explanation, no markdown"* (Orchestrateur)

Cette règle est également appliquée dans le code grâce à une étape de nettoyage automatique. Des expressions régulières suppriment les balises Markdown inutiles (````sql`, ````json`), les préfixes comme *"Cypher :"* ainsi que les commentaires *inline* (`// ...`) avant l'utilisation de la réponse générée.

Cette approche combine des consignes précises dans le prompt et un nettoyage côté code afin de rendre le pipeline plus fiable face aux variations des sorties du modèle.

3.7.2.6 Prompt d'erreur réflexif pour le query rewriting

Le mécanisme de correction itérative repose sur une technique de prompt augmenté par l'erreur : lorsqu'une requête échoue à la validation ou à l'exécution, le message d'erreur précis est réinjecté dans le prompt de l'agent concerné via un `error_hint` :

```
error_hint = f"\n\nPrevious error: {error}. Fix that specific error." if error else ""
```

Pour l'agent Neo4j, ce mécanisme est étendu par une liste de corrections communes directement encodée dans le hint d'erreur :

```
PREVIOUS ERROR - YOU MUST FIX THIS: {error}
Common fixes:
- Check label/relationship spelling against the schema
- Use backticks for special characters: `Label`
- Avoid undefined variables in WHERE/RETURN
- Don't use SQL syntax (no SELECT, FROM, JOIN)
- Use EXISTS { MATCH ... } instead of deprecated WHERE EXISTS()
```

Cette approche transforme chaque échec en signal d'apprentissage local : le modèle dispose non seulement du diagnostic d'erreur, mais aussi d'une taxonomie des causes probables, ce qui augmente significativement le taux de correction dès la première tentative de réécriture.

3.7.2.7 Séparation fait/interprétation dans l'agent Insight

L'agent Insight applique une architecture de prompt en deux temps qui constitue une décision architecturale délibérée contre les hallucinations numériques. Un résumé factuel déterministe est d'abord construit programmatiquement à partir des données brutes (statistiques de max, min, moyenne avec labels associés), puis transmis au modèle avec une instruction explicite :

Your task:

- Write a REAL business insight
- Do NOT simply repeat the raw numbers
- Interpret patterns and trends
- Explain likely business causes
- Sound like a professional analyst
- Keep it concise (4-6 sentences)
- Answer in the SAME language as the user

Cette séparation garantit que les chiffres dans la réponse finale proviennent du calcul Python et non de la génération du modèle, éliminant ainsi le risque principal d'hallucination dans les réponses analytiques.

3.8 Études de cas et résultats préliminaires

Dans cette section, nous présentons une série d'études de cas afin d'évaluer les performances de notre solution de chatbot analytique. Ces études permettent d'analyser la capacité du système à comprendre les requêtes en langage naturel, à générer automatiquement les requêtes appropriées, à produire des visualisations pertinentes ainsi qu'à fournir des analyses métier cohérentes et exploitables.

3.8.1 Étude de Cas 1 : Évolution mensuelle du revenu total en 2022

Scénario

L'objectif de cette étude est d'analyser l'évolution du revenu total de l'entreprise au cours de l'année 2022 afin d'identifier les périodes de croissance et les variations saisonnières.

Requête

Quelle est l'évolution mensuelle du revenu total en 2022 ?

Résultats

Le système produit automatiquement :

- une courbe temporelle illustrant l'évolution mensuelle des revenus .
- un tableau détaillé des valeurs .
- une interprétation métier synthétique.

TAB. 3.2 : Évolution mensuelle du revenu en 2022

Mois	Revenu total
2022-01	1521244386.77
2022-02	1318380720.80
2022-03	1485782207.99
2022-04	1734963318.35
2022-05	1486315354.75
2022-06	1461744176.91
2022-07	1615016099.79
2022-08	1584237345.85
2022-09	1381412830.01
2022-10	1434378222.23
2022-11	1430918765.08
2022-12	1658158857.62

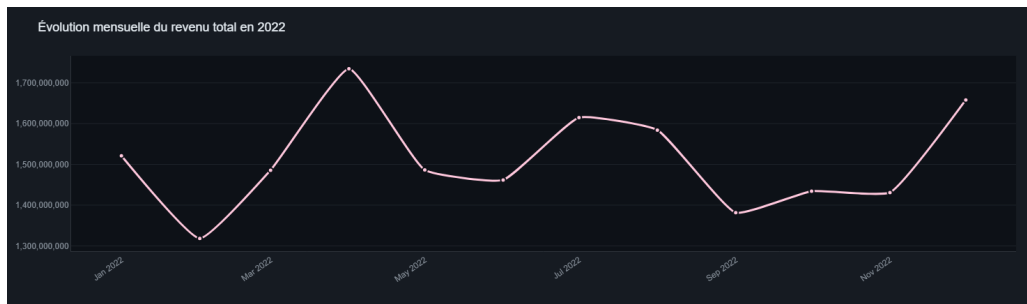


FIG. 3.6 : Graphique de l'évolution mensuelle du revenu en 2022

Analyse métier générée par le système L'évolution mensuelle du revenu total chez Djazzy en 2022 montre une variation significative, avec un minimum observé en février (1,32 Md) et un maximum en avril (1,73 Md), soit une hausse de près de 32%. La courbe présente plusieurs phases de croissance, notamment en avril, en juillet, et en décembre, suggérant une possible saisonnalité des revenus et des pics de consommation à certaines périodes de l'année. Cette dynamique peut être liée à une augmentation de l'usage des services lors de périodes clés de consommation. Elle indique que les revenus de Djazzy ne sont pas stables mais influencés par des cycles de demande. Ainsi, une adaptation des campagnes marketing et des offres promotionnelles sur ces périodes stratégiques pourrait permettre d'optimiser la performance commerciale.

3.8.2 Étude de Cas 2 : Analyse du chiffre d'affaires par wilaya en 2023

Scénario

Cette étude vise à analyser la répartition géographique du chiffre d'affaires afin d'identifier les wilayas générant les revenus les plus élevés et de mieux comprendre les disparités régionales.

Requête

Quel est le chiffre d'affaires de chacune des wilayas en 2023 ?

Résultats

Le système produit automatiquement :

- un histogramme comparatif des revenus par wilaya ;
- un tableau SQL détaillé ;
- une synthèse analytique métier contextualisée.

TAB. 3.3 : Chiffre d'affaires par wilaya en 2023

Wilaya	Chiffre d'affaires
Alger	380712285.37
Annaba	86257811.52
Autres	344329965.70
Batna	86609922.05
Blida	103496694.31
Chlef	69016448.50
Constantine	154745209.89
M'Sila	69065167.00
Oran	223842244.37
Sétif	120706988.06
Tizi Ouzou	86418329.18

Analyse métier générée par le système D'après les données disponibles, il ressort que la wilaya d'Alger détient le chiffre d'affaires le plus élevé en 2023, avec un total de 380,7 millions de dinars. Cette tendance suggère que la densité de population et l'activité commerciale

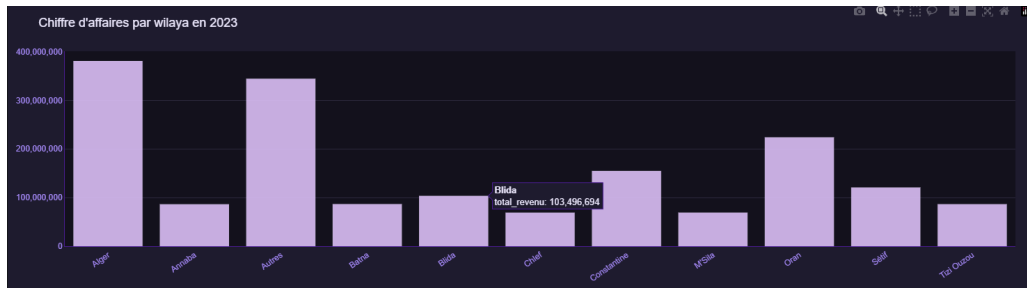


FIG. 3.7 : Graphique du chiffre d'affaires par wilaya en 2023

de la wilaya d'Alger contribuent à sa forte activité de recharge, contrastant avec les autres régions. Les données suggèrent également que les wilayas d'importance régionale ou du littoral, telles que Constantine et Oran, affichent des chiffres d'affaires élevés, ce qui pourrait être lié à leur situation géographique et à leur attractivité économique. Cela souligne l'importance de la présence du réseau dans ces régions pour attirer et retenir les clients.

À l'inverse, certaines wilayas présentent des revenus plus faibles, ce qui peut indiquer un potentiel d'amélioration des infrastructures télécoms et des stratégies commerciales locales.

3.8.3 Conclusion

Les différentes études de cas démontrent que notre système est capable :

- de comprendre des requêtes analytiques formulées en langage naturel
- de générer automatiquement des requêtes SQL pertinentes
- de produire des visualisations interactives adaptées aux données
- de fournir des analyses métier contextualisées et exploitables.

3.9 Évaluation du système Agentic RAG

Cette section présente l'évaluation du système Agentic RAG à travers 30 requêtes réparties en trois niveaux de difficulté. L'objectif est d'analyser les performances des composants, la qualité du raisonnement et les performances globales du système.

3.9.1 Protocole d'évaluation

Pour évaluer notre système Agentic RAG, nous avons préparé 30 requêtes divisées en 3 *batches* de 10 questions. Les requêtes sont classées du niveau simple au niveau complexe afin de voir comment le système réagit selon la difficulté des tâches :

- Le **batch 1** contient des questions simples impliquant une seule table et un seul agent.

- Le **batch 2** contient des requêtes avec plusieurs sources de données et des jointures.
- Le **batch 3** contient les requêtes les plus complexes avec raisonnement *multi-hop*, chaînes d'agents et cas limites.

L'évaluation est réalisée et analysée sur trois couches distinctes :

1. **Couche 1** : Évaluation des composants du système.
2. **Couche 2** : Analyse des trajectoires de raisonnement.
3. **Couche 3** : Performances globales du système.

3.9.2 Batch 1 : Requêtes simples

Le premier *batch* contient des requêtes simples utilisant une seule table sans raisonnement complexe. Le système obtient ici les meilleures performances avec très peu d'erreurs et une latence faible.

- Quel est le nombre total d'abonnés actifs ce mois-ci ?
- Combien de souscriptions au forfait 4G ont été enregistrées aujourd'hui ?
- Quel est le taux de couverture 5G actuel sur le réseau Djezzy ?
- Combien d'antennes sont déployées dans la wilaya d'Alger ?

TAB. 3.4 : Performances globales — Batch 1 (toutes couches)

Couche	Métrique	Moyenne
Couche 1 : Composants		
	Tool Call Success Rate	1,000
	Schema Retrieval Coverage	1,000
	MRR	1,000
	nDCG@5	0,992
	Source Routing Accuracy	1,000
Couche 2 : Trajectoires		
	Task Completion Rate	1,000
	First Attempt Success	100 %
	Retry Rate	0,000
	Step Efficiency	0,898
Couche 3 : Système		
	E2E Latency	6,08 s
	TTFT estimé	1,82 s
	Token Usage	1174
	Empty Result Rate	0,000
	Error Rate	0,000

Les résultats sont excellents sur ce *batch*. Toutes les requêtes sont réussies dès la première tentative sans erreur. La *Step Efficiency* montre que le raisonnement reste très efficace avec peu d'étapes inutiles.

3.9.3 Batch 2 : Requêtes intermédiaires

Le deuxième *batch* contient des requêtes nécessitant plusieurs sources de données, des jointures et parfois de la mémoire contextuelle. Les performances restent bonnes mais une légère baisse apparaît avec l'augmentation de la complexité.

- Comparer le nombre de souscriptions 4G et 5G par région sur les 3 derniers mois.
- Quels sont les wilayas avec le plus fort taux de churn parmi les abonnés ayant souscrit à un forfait data ?
- Donner l'évolution mensuelle du chiffre d'affaires par type d'abonnement (prépayé / post-payé) depuis janvier.

TAB. 3.5 : Performances globales — Batch 2 (toutes couches)

Couche	Métrique	Moyenne
Couche 1 : Composants		
	Tool Call Success Rate	0,982
	Schema Retrieval Coverage	0,965
	MRR	0,943
	nDCG@5	0,909
	Source Routing Accuracy	0,970
Couche 2 : Trajectoires		
	Task Completion Rate	0,900
	First Attempt Success	80 %
	Retry Rate	0,300
	Step Efficiency	0,797
Couche 3 : Système		
	E2E Latency	9,71 s
	TTFT estimé	2,91 s
	Token Usage	2185
	Empty Result Rate	0,100
	Error Rate	0,100

On remarque une augmentation de la latence et de la consommation de *tokens*. Certaines requêtes ont nécessité des *retries* à cause de légers problèmes de *routing* ou de jointures multi-sources.

3.9.4 Batch 3 : Requêtes complexes

Le troisième *batch* contient les requêtes les plus difficiles avec raisonnement *multi-hop*, contradictions latentes dans les données, longues chaînes d'agents et requêtes hors périmètre. Les performances diminuent davantage, ce qui reste logique vu la difficulté des tâches.

- Identifier les zones où la couverture 5G est annoncée comme active mais où aucune souscription 5G n'a été enregistrée, et croiser avec les tickets incidents ouverts sur ces antennes.
- Pourquoi le churn a-t-il augmenté dans le Nord-Est alors que la couverture réseau s'est améliorée sur la même période ?
- « Comparer le churn par wilaya entre le T1 et le T2 en croisant avec le nombre d'antennes actives par zone. »

TAB. 3.6 : Performances globales — Batch 3 (toutes couches)

Couche	Métrique	Moyenne
Couche 1 : Composants		
	Tool Call Success Rate	0,933
	Schema Retrieval Coverage	0,902
	MRR	0,818
	nDCG@5	0,782
	Source Routing Accuracy	0,920
Couche 2 : Trajectoires		
	Task Completion Rate	0,700
	First Attempt Success	40 %
	Retry Rate	1,100
	Step Efficiency	0,698
Couche 3 : Système		
	E2E Latency	16,75 s
	TTFT estimé	5,02 s
	Token Usage	3781
	Empty Result Rate	0,300
	Error Rate	0,300

Les résultats montrent les limites actuelles du système. Les principales difficultés concernent les contradictions au sein des données, les requêtes volontairement hors périmètre et le maintien de la cohérence au long des chaînes d'agents. La latence et la consommation de *tokens* augmentent fortement à cause du nombre élevé de *retries* et du processus de raisonnement mais plus complexe.

3.9.5 Synthèse globale

Le tableau 3.7 regroupe l'ensemble des métadonnées calculées à travers les trois *batches* de test pour offrir une vue comparative globale et exhaustive de l'évolution des performances du système.

TAB. 3.7 : Tableau comparatif et synthèse globale des performances du système

Métrique	Batch 1 (Simple)	Batch 2 (Intermédiaire)	Batch 3 (Complexe)
Task Completion Rate	1,000	0,900	0,700
First Attempt Success	100 %	80 %	40 %
Retry Rate	0,000	0,300	1,100
Step Efficiency	0,898	0,797	0,698
Proxy MRR	1,000	0,943	0,818
Proxy nDCG@5	0,992	0,909	0,782
Schema Retrieval Coverage	1,000	0,965	0,902
Tool Call Success Rate	1,000	0,982	0,933
E2E Latency moyenne	6,08 s	9,71 s	16,75 s
TTFT moyen	1,82 s	2,91 s	5,02 s
Token Usage moyen	1174	2185	3781
Error Rate	0,000	0,100	0,300

3.9.6 Conclusion

Cette évaluation montre que le système Agentic RAG fonctionne parfaitement bien sur les requêtes simples et reste hautement performant sur les requêtes intermédiaires. Les résultats du *batch 1* valident la robustesse du pipeline de base avec un fonctionnement stable et exempt d'erreur. Le *batch 2* et *batch 3* mettent en évidence que les jointures multi-sources et l'orchestration multi-agents augmentent proportionnellement le coût en temps et en ressources de calcul (*tokens*).

On note également que le système préfère échouer proprement via son mécanisme de *fallback* plutôt que de générer de fausses réponses (hallucinations).

3.10 Limites et discussion

Malgré les résultats satisfaisants obtenus lors de l'évaluation, plusieurs limites importantes apparaissent dans notre système Agentic RAG.

3.10.1 Limites identifiées

Performances et latence

Plusieurs étapes sont exécutées successivement (analyse d'intention, récupération du schéma, génération de requête, exécution, synthèse). Le temps de réponse peut augmenter lorsque les requêtes sont complexes ou que les bases contiennent un grand volume de données.

Maintenance

L'ajout de nouvelles sources de données nécessite souvent la création ou l'adaptation d'agents spécialisés. La maintenance devient plus complexe lorsque le nombre de bases augmente.

Coût computationnel

L'utilisation combinée du LLM, du RAG et de plusieurs agents augmente la consommation de ressources. Les coûts d'inférence peuvent devenir importants pour une utilisation intensive.

3.10.2 Perspectives et travaux futurs

Malgré ces limites, le système démontre une bonne robustesse sur les requêtes simples et intermédiaires, et conserve un comportement cohérent même dans les scénarios complexes. Les limites identifiées constituent de solides pistes d'amélioration pour de futurs travaux.

Plusieurs axes d'amélioration peuvent être envisagés :

- **Apprentissage intersession par mémoire à long terme** : Pour améliorer les performances du système sur la durée, nous proposons l'intégration d'une mémoire à long terme persistante entre les sessions. Cette mémoire stocke, pour chaque interaction passée, la question posée par l'utilisateur ainsi que la requête correcte générée (SQL, DSL JSON, Cypher, etc.). Lorsqu'un utilisateur pose une nouvelle question, le système effectue une recherche de similarité dans cette mémoire. Si une question proche a déjà été traitée avec succès, deux scénarios sont possibles :

La question est quasi-identique : la requête correcte est directement réutilisée, ce qui réduit considérablement la latence et évite un raisonnement redondant. La question est similaire sans être identique : la requête passée est injectée comme exemple dans le contexte du modèle, guidant ainsi la génération vers une réponse plus précise et cohérente.

Ce mécanisme transforme chaque interaction réussie en une ressource d'apprentissage pour les suivantes. Au fil du temps, le système devient plus rapide, plus fiable, et mieux adapté aux habitudes et au domaine métier de ses utilisateurs, sans nécessiter de réentraînement du modèle.

- **Apprentissage par renforcement** : L'ajout d'une boucle d'apprentissage par renforcement basée sur les commentaires des utilisateurs (*Reinforcement Learning from User Feedback*). Ces retours permettraient au système d'apprendre progressivement de ses erreurs, d'améliorer ses stratégies de *routing* et d'optimiser dynamiquement les trajectoires de raisonnement au fil des interactions.
- **Approches hybrides** : L'intégration d'approches combinant raisonnement prédéfinis et raisonnement agentique, afin d'améliorer la cohérence logique globale du système tout en conservant la flexibilité et l'adaptabilité des modèles de langage modernes.

Conclusion générale

Ce travail de fin d'études avait pour objectif de concevoir et de développer une architecture Agentic RAG destinée à l'interrogation intelligente de sources de données hétérogènes dans un contexte de Business Intelligence conversationnelle.

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

- **Analyse de l'état de l'art** : étude des limites des approches RAG classiques, des solutions de BI conversationnelle existantes et des architectures Agentic RAG récentes, mettant en évidence leurs difficultés à gérer des environnements multi-sources et des raisonnements complexes.
- **Conception d'une architecture Agentic RAG** : proposition d'une architecture modulaire reposant sur plusieurs agents spécialisés collaborant pour planifier les traitements, sélectionner les sources pertinentes, générer les requêtes adaptées, valider les résultats et produire une réponse analytique cohérente.
- **Développement d'un prototype fonctionnel** : implémentation d'un système intégrant un orchestrateur, des agents spécialisés (SQLite, MongoDB, Neo4j, Elasticsearch), un mécanisme de récupération sémantique des schémas basé sur FAISS, un pipeline de validation des requêtes ainsi que des agents dédiés à la génération de visualisations et d'insights.
- **Évaluation expérimentale** : validation de l'architecture sur plusieurs scénarios de complexité croissante, démontrant sa capacité à traiter des requêtes simples, intermédiaires et multi-sources nécessitant un raisonnement multi-étapes.
- **Analyse des performances** : les résultats obtenus montrent que l'approche proposée fournit des performances satisfaisantes sur la majorité des cas étudiés, tout en mettant en évidence une diminution progressive de la robustesse lorsque la complexité des requêtes et le nombre d'agents impliqués augmentent.
- **Identification des limites** : les expérimentations soulignent plusieurs défis propres aux architectures agentiques, notamment la propagation des erreurs entre agents, l'augmentation de la latence, le coût élevé en consommation de tokens ainsi que la sensibilité aux erreurs de planification ou de génération de requêtes.
- **Perspectives d'amélioration** : plusieurs axes d'évolution peuvent être envisagés, parmi lesquels l'intégration de mécanismes de *Reinforcement Learning* à partir du retour utilisateur, le développement de stratégies de vérification inter-agents plus avancées, l'optimisation des coûts d'inférence, l'amélioration des mécanismes de récupération sémantique et l'utilisation d'approches hybrides combinant raisonnement déterministe et agents LLM.

En définitive, ce travail met en évidence le potentiel des architectures Agentic RAG pour dépasser les limites des systèmes RAG traditionnels dans le domaine de la Business Intelligence conversationnelle. En associant récupération sémantique, planification, raisonnement et collaboration entre agents spécialisés, l'architecture proposée constitue une base solide pour le

développement de systèmes d'aide à la décision capables d'interroger intelligemment des environnements de données d'entreprise complexes et hétérogènes. Elle ouvre ainsi des perspectives prometteuses pour les futures générations de systèmes analytiques autonomes.

Bibliographie

- [1] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled : A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley, 2012.
- [2] William H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, 1992.
- [3] Ashish Vaswani et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.
- [4] Aditi Singh, Abul Ehtesham, Saket Kumar, Tala Talaei Khoei, and Athanasios V Vasilakos. Agentic retrieval-augmented generation : A survey on agentic rag. 2026.
- [5] Yunfan Gao, Yun Xiong, Meng Wang, et al. Modular rag : Transforming rag systems into lego-like reconfigurable frameworks. 2024.
- [6] Sok : Agentic retrieval-augmented generation (rag) : Taxonomy, architectures, evaluation, and research directions. 2026.
- [7] Carlo Verellis. *Business Intelligence : Data Mining and Optimization for Decision Making*. John Wiley & Sons, 2009.
- [8] Joe Reis and Matt Housley. *Fundamentals of Data Engineering : Plan and Build Robust Data Systems*. O'Reilly Media, 2022.
- [9] Ralph Kimball. *The Data Warehouse Toolkit*. John Wiley & Sons, 1996.
- [10] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015.
- [13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8) :1735–1780, 1997.
- [14] Ian Goodfellow et al. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.
- [15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert : Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.
- [16] Tom B. Brown et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

- [17] Alexey Dosovitskiy et al. An image is worth 16x16 words : Transformers for image recognition at scale. In *ICLR*, 2021.
- [18] Rishi Bommasani et al. On the opportunities and risks of foundation models. Technical report, Stanford CRFM, 2021.
- [19] Wayne Xin Zhao et al. A survey of large language models. *arXiv preprint arXiv :2303.18223*, 2023.
- [20] Jason Wei et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35, 2022.
- [21] Yunfan Gao et al. Retrieval-augmented generation for large language models : A survey. *arXiv preprint*, 2024.
- [22] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. *arXiv preprint arXiv :2212.10496*, 2022.
- [23] Jintao Liang, Gang Su, Huifeng Lin, You Wu, Rui Zhao, and Ziyue Li. Reasoning rag via system 1 or system 2: A survey on reasoning agentic retrieval-augmented generation for industry challenges. *arXiv preprint*, 2025.
- [24] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. Raptor : Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations (ICLR)*, 2024.
- [25] Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, et al. Memory in the age of ai agents : A survey forms, functions and dynamics. *arXiv preprint*, 2026.
- [26] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [27] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React : Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [28] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts : Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [29] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu : A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, 2002.
- [30] Chin-Yew Lin. Rouge : A package for automatic evaluation of summaries. In *ACL Workshop on Text Summarization Branches Out*, pages 74–81, 2004.
- [31] Pengfei Liu et al. Pre-train, prompt, and predict : A systematic survey of prompting methods in nlp. *ACM Computing Surveys*, 55(9) :1–35, 2023.
- [32] Yongchao Zhou et al. Large language models are human-level prompt engineers. In *ICLR*, 2023.

- [33] Yao Lu et al. Fantastically ordered prompts and where to find them. In *ACL*, pages 8086–8098, 2022.
- [34] Takeshi Kojima et al. Large language models are zero-shot reasoners. *Advances in Neural Information Processing Systems*, 35, 2022.
- [35] Shunyu Yao et al. Tree of thoughts : Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2023.
- [36] Xuezhi Wang et al. Self-consistency improves chain of thought reasoning in language models. In *ICLR*, 2023.
- [37] Shunyu Yao et al. React : Synergizing reasoning and acting in language models. In *ICLR*, 2023.
- [38] Sewon Min et al. Rethinking the role of demonstrations : What makes in-context learning work ? In *EMNLP*, 2022.
- [39] Jules White et al. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv :2302.11382*, 2023.
- [40] Nelson F. Liu et al. Lost in the middle : How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2023.

Appendices

.1 Ingénierie des prompts (Prompt Engineering)

.1.1 Définition et importance

L'ingénierie des prompts désigne l'ensemble des techniques visant à concevoir, structurer et optimiser les instructions textuelles (*prompts*) soumises à un LLM afin d'orienter ses sorties vers un résultat souhaité. Contrairement au *fine-tuning*, qui modifie les poids du modèle, l'ingénierie des prompts agit exclusivement sur l'entrée, sans altérer les paramètres internes du modèle [31].

L'importance de cette discipline tient à un constat fondamental : pour un même modèle et une même tâche, la qualité de la réponse varie considérablement selon la formulation du prompt. Un prompt mal conçu peut produire des réponses vagues, incorrectes ou hors sujet, tandis qu'un prompt bien structuré permet d'exploiter pleinement les capacités du modèle [32]. L'ingénierie des prompts est ainsi devenue une compétence clé dans le déploiement opérationnel des LLMs et un champ de recherche à part entière, à l'intersection du traitement du langage naturel et de l'interaction homme-machine.

.1.2 Taxonomie des techniques de prompting

- a) **Zero-shot prompting.** Le *zero-shot prompting* consiste à soumettre une instruction au modèle sans fournir aucun exemple de démonstration. Le modèle s'appuie alors uniquement sur ses connaissances acquises durant le pré-entraînement pour générer une réponse. Cette approche exploite la capacité de généralisation des LLMs mais peut s'avérer insuffisante pour des tâches complexes ou peu représentées dans les données d'entraînement [16]. Un exemple typique serait : « Traduis le texte suivant en anglais : [...] », sans aucun exemple préalable.
- b) **Few-shot prompting.** Le *few-shot prompting* [16] enrichit le prompt avec un petit nombre d'exemples illustrant le format entrée-sortie attendu. Ces exemples, appelés *demonstrations*, agissent comme un conditionnement implicite qui oriente le modèle vers la structure de réponse souhaitée. Notons que la sélection et l'ordre des exemples influencent significativement les résultats, un phénomène documenté comme *order sensitivity* [33].
- c) **Chain-of-Thought (CoT) prompting.** Le *chain-of-thought prompting* [20] constitue l'une des avancées les plus importantes de l'ingénierie des prompts. L'idée centrale est d'encourager le modèle à expliciter une chaîne de raisonnement intermédiaire avant de formuler sa réponse finale. En incluant dans le prompt des exemples annotés avec des étapes de raisonnement détaillées (—) ou simplement en ajoutant la directive « Réfléchis étape par étape » (*zero-shot CoT* [34]) on observe une amélioration substantielle des performances sur les tâches arithmétiques, de logique et de raisonnement en plusieurs étapes.
- d) **Tree-of-Thoughts (ToT).** Le *Tree-of-Thoughts* [35] généralise le CoT en permettant au modèle d'explorer simultanément plusieurs chemins de raisonnement sous forme d'arbre, d'évaluer chaque branche et de revenir en arrière si une voie s'avère non productive. Cette approche, inspirée des méthodes de recherche heuristique (BFS, DFS), améliore considérablement les performances sur des problèmes nécessitant de la planification ou de la créativité structurée.

- e) **Self-consistency.** La technique de *self-consistency* [36] complète le CoT en générant plusieurs chaînes de raisonnement indépendantes pour un même problème, puis en sélectionnant la réponse finale par vote majoritaire. Ce mécanisme d'ensemble réduit la variabilité des réponses et améliore la robustesse, notamment sur les tâches de raisonnement mathématique et de bon sens.
- f) **ReAct (Reasoning + Acting).** Le paradigme *ReAct* [37] intègre raisonnement et action au sein d'un même prompt : le modèle alterne entre des étapes de réflexion interne et des étapes d'action (appel à des outils externes tels que des moteurs de recherche ou des bases de données). Cette architecture constitue le socle des systèmes d'agents LLM autonomes, capables d'interagir dynamiquement avec leur environnement pour accomplir des tâches complexes à plusieurs tours.

.1.3 Optimisation automatique des prompts

Face à la difficulté de concevoir manuellement des prompts optimaux, plusieurs approches d'optimisation automatique ont été proposées :

- **Automatic Prompt Engineer (APE) :** Ce cadre [19] exploite un LLM pour générer et évaluer automatiquement des candidats-prompts pour une tâche donnée. Le prompt le plus performant est sélectionné par une procédure de recherche guidée par le score obtenu sur un ensemble de validation. APE surpasse les prompts conçus manuellement sur plusieurs benchmarks, démontrant le potentiel de la génération automatique de prompts.
- **Prompt tuning et prefix tuning :** À mi-chemin entre l'ingénierie des prompts et le *fine-tuning*, le *prompt tuning* [25] et le *prefix tuning* [26] consistent à apprendre des vecteurs continus (*soft prompts*) dans l'espace des embeddings du modèle, en maintenant les poids du LLM gelés. Ces vecteurs sont optimisés par gradient sur une tâche cible, offrant une alternative légère au *fine-tuning* complet tout en conservant la flexibilité des modèles pré-entraînés.

.1.4 Bonnes pratiques et principes de conception

La littérature et la pratique industrielle convergent vers plusieurs principes guidant la conception de prompts efficaces :

- **Clarté et précision :** Un prompt doit formuler la tâche de manière non ambiguë, en spécifiant le rôle attendu du modèle (*role prompting* : « Tu es un expert ... »), le format de sortie souhaité (JSON, liste, paragraphe) et les contraintes éventuelles (longueur, niveau de langue, public cible).
- **Décomposition des tâches :** Pour les tâches complexes, il est recommandé de les décomposer en sous-tâches séquentielles plutôt que de soumettre une instruction monolithique. Cette approche, cohérente avec le paradigme CoT, réduit la charge cognitive imposée au modèle et limite les erreurs de propagation [20].

- **Exemples positifs et négatifs** : Fournir à la fois des exemples de bonnes et de mauvaises réponses (*contrastive prompting*) aide le modèle à mieux cerner les frontières de la tâche et à éviter les erreurs fréquentes [38].
- **Contrôle du format de sortie** : L'inclusion d'instructions explicites sur la structure de sortie — via des balises XML, du Markdown structuré ou un schéma JSON — améliore la fiabilité et la parseabilité des réponses dans les pipelines automatisés [39].
- **Gestion du contexte** : Les informations les plus pertinentes doivent être placées en début ou en fin de prompt : les LLMs présentent un biais dit de *lost-in-the-middle*, les informations situées au centre d'un long contexte étant moins bien mémorisées [40].

.2 Architecture et Choix Technologiques

Cette section présente les principaux choix technologiques du système, notamment le modèle de LLM, le framework d'orchestration multi-agents et le modèle d'embedding.

.2.1 Large Language Model (LLM)

Le choix du LLM constitue la décision architecturale la plus structurante du système, car il conditionne directement la précision de la génération des requêtes, l'efficacité de l'orchestration et la pertinence des *insights* produits.

TAB. 8 : Espace des alternatives pour le choix du LLM

Type de Modèle	Fournisseur / Éditeur	Modèle LLM	Fenêtre de Contexte
Local	Mistral AI	Mistral 7B	8 192 tokens
Local	HuggingFace (Fine-tuned)	Zephyr 7B	8 192 tokens
Local	Microsoft	Phi-2	2 048 tokens
Local	Meta	LLaMA 3.1 8B Instruct	128 000 tokens
API-Based	OpenAI	GPT-4	128 000 tokens
API-Based	OpenAI	GPT-4 Turbo	128 000 tokens
API-Based	OpenAI	GPT-3.5 Turbo	16 385 tokens
API-Based	Google	Gemini 1.5 Flash	1 048 576 tokens
API-Based	Google	Gemini 1.5 Pro	2 097 152 tokens

Espace des alternatives (Modèles LLM) Le choix du LLM constitue une décision architecturale critique dans le système proposé pour *Djezzy*, car il influence directement :

- la qualité de génération des requêtes (SQL, Cypher, MongoDB)

- la capacité d'orchestration multi-agents
- la pertinence des analyses et des insights produits

Dans un contexte télécom comme celui de **Djezzy**, ce choix est particulièrement sensible en raison de :

- la confidentialité des données (KPIs réseau, données clients, métriques internes)
- les contraintes de coût à grande échelle
- la nécessité d'un fonctionnement potentiellement on-premise

Les solutions basées sur API (GPT-4, Gemini) offrent des performances élevées et de très grandes fenêtres de contexte. Cependant, dans le cadre de **Djezzy**, elles présentent des limites majeures :

- exposition potentielle de données sensibles vers des services externes
- coûts élevés liés aux appels API à grande échelle
- dépendance forte à une connectivité externe

Les modèles locaux représentent une alternative plus adaptée au contexte industriel de Djezzy, mais avec des compromis sur la taille du contexte et la robustesse. Par exemple, *Phi-2* a été écarté directement en raison de sa fenêtre de contexte insuffisante pour gérer simultanément les prompts système, les schémas de bases de données et l'historique conversationnel.

Choix retenu : LLaMA 3.1 8B Instruct Le modèle `meta-llama/Llama-3.1-8B-Instruct` a été retenu pour le système de Djezzy pour les raisons suivantes :

- **Capacité de contexte** : 128 000 tokens, suffisants pour intégrer les chunks de schémas issus du RAG, les instructions des agents et le contexte utilisateur.
- **Suivi d'instructions** : modèle optimisé pour le *instruction tuning*, ce qui améliore fortement la génération de requêtes structurées (SQL, Cypher, MongoDB).
- **Adaptation au système multi-agents** : bonne stabilité dans des pipelines où plusieurs agents spécialisés interviennent (orchestrateur, validator, query generator).
- **Compatibilité déploiement local** : permet une intégration on-premise, alignée avec les contraintes de sécurité de Djezzy.

.2.2 Framework d'Orchestration Multi-Agents

La couche d'orchestration représente le squelette du système : elle régit la communication entre les agents, le transit des états et l'exécution des décisions conditionnelles.

Espace des alternatives (Frameworks d’orchestration) Le tableau 9 compare les paradigmes des principaux environnements de développement multi-agents du marché.

TAB. 9 : Espace des alternatives pour les frameworks d’orchestration

Framework	Paradigme	Gestion d’État	Flux Conditionnel	Maturité
LangChain Chains	Séquentiel linéaire	Implicite	Limité	Très mature
LangChain Agents	Boucle ReAct	Implicite	Via tool-calling	Mature
LangGraph	Graphe d’états finis	Explicite (TypedDict)	Natif (Routeurs)	Récent mais stable
AutoGen	Multi-agents conv.	Partagé	Échange messages	Émergent
CrewAI	Approche par rôles	Partagé	Séquentiel / Parallèle	Récent
Semantic Kernel	Basé sur plug-ins	Partagé	Planificateur auto.	Mature

Les approches linéaires comme *LangChain Chains* s’avèrent inadaptées face aux exigences de notre système, qui requiert nativement des boucles de correction (*retry*), des branchements conditionnels complexes et une persistance d’état structurée. Bien que *LangChain Agents* (ReAct) s’en rapproche, l’opacité de son flux de contrôle introduit un non-déterminisme difficile à maîtriser en environnement de production.

D’un autre côté, les frameworks purement conversationnels tels qu’*AutoGen* ou *CrewAI* excellent dans les tâches créatives ou ouvertes, mais manquent de rigueur pour un pipeline analytique. La verbosité et l’imprévisibilité de leurs échanges inter-agents compromettent la traçabilité et la précision indispensables à la génération de requêtes de bases de données.

Choix retenu : LangGraph *LangGraph* s’est imposé comme l’unique solution capable de concilier flexibilité et déterminisme. L’architecture repose sur une gestion d’état explicite et typée via un `TypedDict` partagé.

Chaque transition du pipeline est gouvernée par un routeur conditionnel prédictible, ce qui permet de modéliser naturellement la boucle de validation critique (`validator` → `retry` → `query_generator` → `validator`). De plus, l’orientation “graphe” permet de traiter le routage des intentions (`DATA_QUERY` vs `CONVERSATIONAL`) ou la gestion des anomalies comme des nœuds de première classe. Chaque nœud opérant comme une fonction Python pure, le débogage et l’écriture de tests unitaires isolés s’en trouvent grandement facilités.

.2.3 Modèle d’Embedding et Index Vectoriel

Le module Schema RAG s’articule autour de deux briques distinctes : le modèle d’embedding (vectorisation textuelle) et l’index vectoriel (recherche par similarité).

Choix retenus Le couple `all-mpnet-base-v2` et *FAISS HNSW* a été désigné comme l’architecture cible optimale :

- **Pour l'Embedding** : La dimensionnalité de 768 d'`all-mpnet-base-v2` fournit une richesse sémantique fine, cruciale pour discriminer des propriétés aux intitulés proches (ex : `total_amount` vs `total_recharges_amount`). Sa fenêtre de 384 tokens enveloppe confortablement les descriptions structurelles de nos tables.
- **Pour l'Indexation** : *FAISS HNSW* concilie une recherche ultra-rapide en $\mathcal{O}(\log N)$ et une excellente exactitude (96 % à 99 %), sans exiger d'infrastructure tierce. Contrairement aux approches de type *IVF* qui nécessitent une phase d'entraînement préalable sur un corpus figé, *HNSW* est immédiatement opérationnel et supporte nativement l'évolution dynamique des schémas.