Utilisation des modèles de langages LLMs pour l'assistance dans les tâches en intelligence d'affaires

WILHELMY Felix
Département de génie LOG & TI
École de Technologie Supérieure
Montréal, Canada
felix.wilhelmy.1@ens.etsmtl.ca

NOUBISSI KOM Carmen Wilfred
Département de génie LOG & TI
École de Technologie Supérieure
Montréal, Canada
carmen-wilfred.noubissi-kom.1@ens.etsmtl.ca

LAAZIRI Othman
Département de génie LOG & TI
École de Technologie Supérieure
Montréal, Canada
Othman.laaziri.1@ens.etsmtl.ca

***Résumé*—Les systèmes d'intelligence d'affaires (Business Intelligence, BI) sont essentiels aux organisations modernes pour transformer de grandes quantités de données en informations exploitables facilitant la prise de décisions stratégiques. Cependant, les méthodes traditionnelles de BI peinent à gérer efficacement la complexité et l'évolution constante des besoins organisationnels. Cet article examine comment les récents progrès en intelligence artificielle (IA), et en particulier les *Large Language Models* (LLMs), peuvent répondre à ces défis. Dans notre discussion, nous proposons un prototype de système qui utilise ces technologies et qui est adapté aux besoins spécifiques de l'entreprise Forester Inc.**

Index Terms*—intelligence artificielle, intelligence d'affaires, *Large Language Models

I. Introduction

Fondée en 1950 dans le sud du Québec, Forester est une entreprise commerciale diversifiée opérant dans plusieurs secteurs, dont la vente de matériel informatique et industriel, ainsi que l'édition et la commercialisation de solutions logicielles. Confrontée à un environnement commercial de plus en plus compétitif, Forester cherche aujourd'hui à améliorer son efficacité opérationnelle, sa communication interne et externe, ainsi que la satisfaction de ses employés et de ses clients. Pour cela, elle souhaite répondre rapidement et précisément aux demandes, automatiser les tâches répétitives, assurer un support continu (24/7) et optimiser la gestion de l'information en analysant efficacement des données provenant de multiples sources afin de mieux identifier opportunités et risques stratégiques.

Malgré leur importance, les systèmes traditionnels d'intelligence d'affaires (BI) demandent beaucoup de temps et de ressources, et sont souvent sujets à des erreurs : difficulté à saisir précisément les besoins réels des utilisateurs, inefficacité dans la gestion des analyses complexes et manque de collaboration entre de multiples parties prenantes. Ces dernières années, les avancées impressionnantes en intelligence artificielle, notamment les LLMs et l'IA générative, représentent une opportunité prometteuse pour surmonter ces défis.

Cet article propose une revue de littérature structurée visant à explorer l'intégration des LLMs dans les systèmes BI via des interfaces conversationnelles. La section I détaille les concepts fondamentaux nécessaires à la compréhension de cette revue ; la section II explore des études de cas d'usage comme SiriusBI [1] et AutoBIR [2], où des architectures LLM ont été mises en œuvre avec succès dans des environnements BI réels. La section III poursuit avec une discussion et la proposition d'un prototype de chatbot conversationnel spécifiquement conçu pour Forester Inc. Enfin, la section IV conclut en synthétisant ce qui a été vu et la faisabilité d'un tel projet.

II. Concepts fondamentaux

A. Intelligence d'affaires : des données aux décisions

La Business Intelligence (BI) regroupe les méthodes, outils et pratiques visant à collecter, stocker, analyser et présenter des données pour aider à la prise de décision. Un système BI classique se compose généralement des étapes suivantes :

1) **Extraction et intégration** des données (ETL) depuis des sources variées (bases de données relationnelles, fichiers, ERP, CRM, etc.) ;
2) **Stockage** des données dans un entrepôt ou un data lake ;
3) **Modélisation** et agrégation des données pour créer des indicateurs clés de performance (KPI) ;
4) **Analyse** via des requêtes SQL, des scripts analytiques (Python, R) ou des solutions de BI (tableaux de bord, rapports, cubes OLAP) ;
5) **Restitution** sous forme de tableaux de bord, graphiques interactifs ou rapports automatisés pour les décideurs.

Ces étapes requièrent souvent un certain niveau d'expertise technique : il faut savoir rédiger des requêtes complexes, comprendre la structure des entrepôts de données et interpréter les résultats. Par conséquent, les utilisateurs métiers (non-développeurs) ont tendance à dépendre des équipes techniques pour formuler des analyses, ce qui engendre des délais et des risques de mauvaise interprétation.

B. Les Larges Modèles de Langages (LLMs)

La modélisation du langage, aussi appelée *Natural Language Processing* (NLP), est un sujet étudié depuis longtemps par des chercheurs tels que Shannon (1950) dans l'application de la théorie de l'information au langage humain.

Ces systèmes peuvent être catégorisés en quatre grands groupes [3] en fonction de leur approche et de leur évolution :

— **SLMs** (*Statistical Language Models*) : ils voient le texte comme une séquence de mots et estiment la probabilité de cette séquence en multipliant les probabilités individuelles de chaque mot selon un modèle statistique.
— **NLMs** (*Neural Language Models*) : ils gèrent la rareté des données en mappant les mots sur des vecteurs continus de faible dimension et prédire le mot suivant à partir de l'agrégation des vecteurs d'intégration des mots précédents au moyen de réseaux neuronaux.
— **PLMs** (*Pre-trained Language Models*) : l'apprentissage et l'inférence suivent le paradigme du pré-apprentissage et du réglage fin, où des modèles de langage utilisant des réseaux de neurones récurrents ou des transformers sont pré-entraînés sur des corpus de textes non étiquetés à l'échelle du Web, puis ajustés sur de petites quantités de données étiquetées pour des tâches spécifiques.
— **LLMs** (*Large Language Models*) : il s'agit principalement de modèles linguistiques neuronaux basés sur des transformers, contenant des dizaines à des centaines de milliards de paramètres pré-entraînés sur des données textuelles massives (par exemple : PaLM, LLaMA, GPT-4).

Les avancées récentes en matière de modèles de langage de grande taille, basés sur des transformers entraînés à l'échelle du Web, ont considérablement amélioré les capacités des modèles de langage. Par exemple, on peut citer *ChatGPT* et *GPT-4* d'OpenAI, qui peuvent être utilisés non seulement pour le traitement du langage naturel, mais aussi comme solveurs de tâches générales alimentant des systèmes tels que Microsoft Copilot. Ils peuvent suivre des instructions humaines pour des tâches complexes en effectuant un raisonnement en plusieurs étapes si nécessaire.

C. Les architectures transformers

Les *Transformers*, introduits par Vaswani et al. (2017) [4], reposent sur un mécanisme d'auto-attention multi-têtes qui permet de modéliser efficacement les dépendances globales au sein d'une séquence de données. Grâce à cette architecture, on peut paralléliser l'entraînement à grande échelle et capturer des dépendances à longue portée, ce qui a propulsé l'émergence des LLMs tels que nous les connaissons aujourd'hui. L'architecture a notamment été popularisée grâce à OpenAI et au célèbre ChatGPT [5].

Les transformers, de manière générale, sont composés de deux parties : un encodeur et un décodeur. Toutefois, la plupart des LLMs implémentent uniquement la partie décodeur. Ces systèmes analysent le sens d'une phrase et génèrent l'élément suivant dans la séquence, un élément à la fois. À chaque étape, le modèle autorégressif consomme les symboles générés précédemment comme entrée supplémentaire.

Concrètement, pour une séquence d'entrée $X \in \mathbb{R}^{n \times d_{\text{model}}}$, on définit

$$Q = X\,W_Q, \quad K = X\,W_K, \quad V = X\,W_V \tag{1}$$

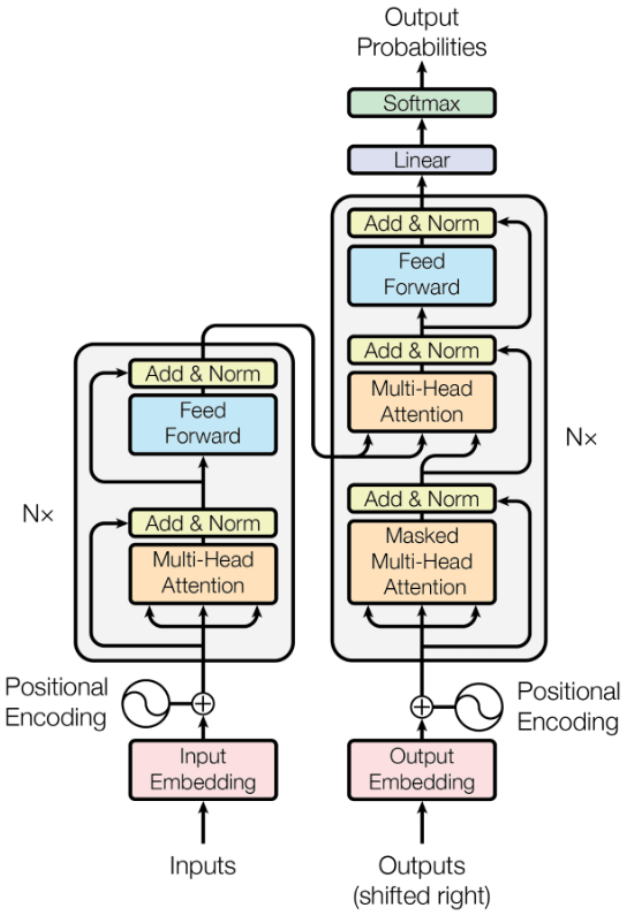

FIGURE 1. Architecture d'un Transformer. À gauche, l'encodeur est constitué de couches d'auto-attention multi-têtes suivies de réseaux feed-forward, chacune entourée d'une connexion résiduelle et d'une normalisation de couche. À droite, le décodeur adopte la même structure, avec en plus un mécanisme d'attention croisée sur la sortie de l'encodeur et un masque temporel pour garantir que la prédiction à la position i ne dépend que des positions $< i$.

avec $W_Q, W_K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ et $W_V \in \mathbb{R}^{d_{\text{model}} \times d_v}$. On calcule ensuite les scores

$$S = \frac{Q\,K^{\mathsf{T}}}{\sqrt{d_k}}, \quad A = \text{softmax}(S) \tag{2}$$

puis la sortie de l'auto-attention s'écrit

$$\text{Attention}(Q, K, V) \;=\; A\,V \tag{3}$$

Chaque vecteur de sortie est ainsi une combinaison convexe des lignes de V, pondérée par la similarité requête/clé.

— **Encodeur :** l'encodeur [4] est composé d'une pile de couches identiques. Chaque couche comporte deux sous-couches. La première est un mécanisme d'auto-attention multi-têtes, et la seconde est un réseau feed-forward positionnel. On utilise une connexion résiduelle autour de chacune des deux sous-couches, suivie d'une normalisation de couche. Autrement dit, la sortie de chaque sous-couche est :

$$\text{LayerNorm}\big(x + \text{Sublayer}(x)\big) \tag{4}$$

où $\mathrm{Sublayer}(x)$ est la fonction implémentée par la sous-couche elle-même. Pour faciliter ces connexions résiduelles, toutes les sous-couches du modèle, ainsi que les couches d'intégration, produisent des sorties de même dimension, à savoir d_{model}.

— **Décodeur :** le décodeur [4] est également composé d'une pile de couches identiques. Outre les deux sous-couches de chaque couche d'encodeur, le décodeur insère une troisième sous-couche qui effectue une attention multi-têtes sur la sortie de la pile d'encodeur. Comme pour l'encodeur, on utilise des connexions résiduelles autour de chaque sous-couche, puis une normalisation de couche. La sous-couche d'auto-attention du décodeur est masquée afin d'empêcher les positions de se concentrer sur les positions suivantes. Ce masquage, combiné au décalage d'une position des embeddings de sortie, garantit que les prédictions pour la position i ne peuvent dépendre que des sorties connues aux positions inférieures à i.

III. Utilisation des LLMs en BI

Les systèmes d'intelligence d'affaires sont en constante évolution vers des outils à la fois plus intuitifs et plus puissants. Une avenue explorée récemment s'appuie sur l'intégration des modèles de langage de grande taille (LLMs), qui permettent de poser des questions complexes en langage naturel et d'obtenir des analyses ou des requêtes sans compétence technique avancée. Cela facilite l'accès à l'analytique et implique davantage les utilisateurs métiers dans les processus décisionnels.

A. Structurer les échanges entre modules

Une solution BI performante repose sur un enchaînement de modules spécialisés : analyse de la requête en langage naturel, génération de code (SQL ou Python), exécution sur la base de données, puis restitution visuelle ou textuelle.

Il est crucial d'organiser clairement les échanges entre ces modules. SiriusBI, par exemple, utilise le module MRD-Q (*Multi-Round Dialogue with Querying*) pour clarifier les requêtes floues. DataLab adopte, quant à elle, une architecture multi-agents à spécialisation définie.

B. Intégrer les spécificités du domaine

Dans un contexte réel, une application BI requiert une quantité considérable d'informations propres au domaine concerné. Pour fonctionner correctement, un LLM doit comprendre la structure, les règles et les indicateurs spécifiques à l'organisation.

C. Compréhension et analyse des données

Pour un *ChatBI* (évolution du système BI traditionnel vers un système capable de comprendre des requêtes sous forme de langage naturel [1]), il ne suffit pas de connaître le langage naturel : il faut aussi savoir connecter la requête d'un utilisateur aux données de l'entreprise. Cela demande une connaissance de la structure, des indicateurs utilisés et de leur interprétation. Il est également crucial d'avoir un système agnostique capable d'agréger différentes sources de données.

Plusieurs approches récentes, comme celle de Jiang et al. (2024) [1] dans le système SiriusBI, montrent l'intérêt de créer un graphe de connaissances qui rassemble ces informations à partir de sources internes (bases de données, scripts SQL/Python, dictionnaires métier). Ce graphe guide alors le modèle pour qu'il interprète correctement les questions, même si elles sont formulées de façon floue ou incomplète.

D. Gestion de la cohérence des résultats

La cohérence des réponses d'un ChatBI est cruciale et loin d'être garantie. Plusieurs techniques ont été proposées pour améliorer cet aspect :

— Utilisation d'un historique conversationnel [1], ce qui permet au ChatBI d'améliorer grandement sa compréhension du domaine et des requêtes antérieures de l'entreprise ;

— Techniques d'apprentissage auto-supervisées via des systèmes de correction du code généré par le système [2] ;

— Dépendance entre les cellules pour reconstruire le raisonnement analytique.

Une stratégie mixte selon le mode d'interaction (conversationnel ou code) pourrait offrir davantage de robustesse.

E. Offrir une interface adaptée à tous les profils

Une interface bien conçue favorise l'adoption : elle doit permettre à des utilisateurs de différents niveaux d'interagir efficacement avec l'outil selon leurs besoins. Que ce soit via une interface unifiée ou intégrée dans des environnements existants (notebooks, outils de BI), l'objectif est de réduire la friction, accompagner l'utilisateur et faciliter la collaboration.

F. Considérations de sécurité

Tel que souligné par Busany et al. (2024) [2], un enjeu majeur dans une telle implémentation est la sécurité des données. Un tel système pourrait générer des requêtes susceptibles de compromettre des données privées si mal configuré ; c'est pourquoi il est important d'implémenter des mécanismes d'anonymisation des données et d'éviter l'usage de données à caractère personnel si l'on mettait en œuvre un tel système. De plus, un employé malveillant pourrait obtenir un accès supérieur à ses propres droits grâce au prompt engineering.

IV. Architecture d'un système ChatBI

Un système ChatBI repose sur quatre modules interdépendants qui transforment une requête en langage naturel en analyses exploitables. Le *Module de dialogue* établit et maintient la conversation, clarifie les intentions et s'assure que la demande est complète. Le *Module de génération de requêtes* traduit cette intention en requêtes SQL (ou équivalent) conformes aux schémas et règles de l'entreprise. Le *Module Insight* exécute ces requêtes, produit les artefacts attendus (visualisations, rapports, statistiques) et offre des analyses complémentaires. Enfin, la *Base de connaissances* centralise

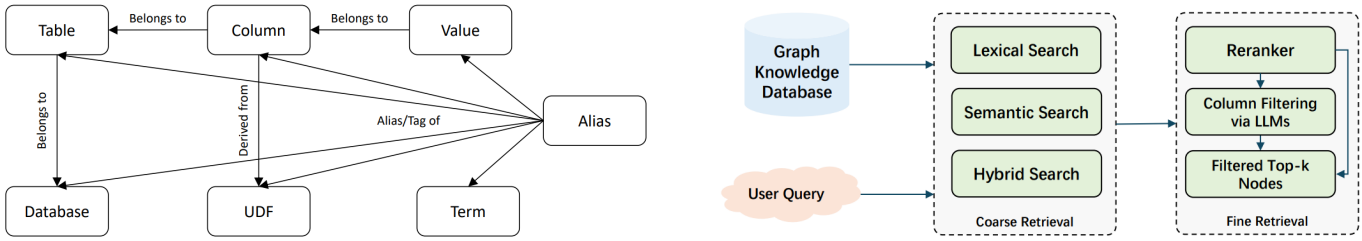

FIGURE 2. Intégration des connaissances et recherche dans un système BI assisté par LLM. *à gauche*, le *graphe de connaissances (Knowledge Graph)* organise les tables physiques, les définitions métier et les relations sémantiques sous forme de triplets (nœuds = entités, arêtes = relations). *À droite*, le module de recherche assisté par LLM interroge ce graphe pour formuler, raffiner et valider des requêtes en langage naturel, en combinant la précision factuelle du graphe et la générativité du LLM pour améliorer la pertinence des réponses.

toutes les métadonnées, définitions métier et règles de sécurité permettant de guider les autres modules et de garantir la validité des résultats.

A. Module de dialogue

Ce module reçoit la requête en langage naturel et entame un dialogue en plusieurs étapes afin de vérifier la syntaxe, détecter les entités métier (dimensions, indicateurs) et compléter les informations à partir de l'historique. Si un détail manque, il interroge l'utilisateur. Ensuite, il consulte la base de connaissances pour valider les termes métier (tables, champs, synonymes). Enfin, il synthétise la requête interprétée et la soumet à l'utilisateur pour confirmation. Cette approche garantit la précision de l'intention sans exiger de compétences techniques.

B. Module de requêtes

Une fois l'intention validée, ce module identifie les tables et jointures nécessaires via la base de connaissances, puis construit la requête SQL (ou API équivalente) selon les conventions internes. Pour assurer la fiabilité, il crée des tests unitaires. Si un test échoue, la requête est ajustée et retestée jusqu'à obtenir un résultat cohérent. En mode supervisé, l'utilisateur peut valider chaque étape (sélection des tables, filtres, structure de la requête) avant exécution.

C. Module Insight

Après exécution, ce module récupère les résultats bruts pour produire :

- des *visualisations* : graphiques, tableaux de bord ;
- des *rapports analytiques* : résumés textuels, PDF, exports ;
- d'autres *livrables* : CSV, modèles statistiques, etc.

L'utilisateur visualise d'abord une version préliminaire, puis peut demander des ajustements (changement de graphique, affinement de filtres, analyses complémentaires). Le processus peut alors repartir d'un nouveau dialogue pour affiner le paramétrage.

D. Base de connaissances et modélisation

La base de connaissances est le référentiel central : un *graphe de connaissances* relie les tables physiques aux définitions logiques (termes métier, abréviations, relations hiérarchiques). Elle contient également :

- le *glossaire métier* : définitions des indicateurs clés et synonymes internes ;
- les *métadonnées techniques* : schémas, types, contraintes, règles d'accès.

Ce référentiel permet de traduire une requête naturelle en entités de la base de données, d'éviter les ambiguïtés et de guider chaque module dans la génération et l'exécution des requêtes.

V. ÉTUDES DE CAS ET RÉSULTATS OBSERVÉS

A. SiriusBI (Jiang et al., 2024)

Jiang et al. (2024) présentent SiriusBI, un système NL2SQL capable de gérer les requêtes single-round et multi-round. Leur approche combine un *Knowledge Graph* pour modéliser la structure de la base et les définitions métier, et le module MRD-Q (*Multi-Round Dialogue with Querying*) pour clarifier l'intention avant de générer la requête SQL. La version « two-step » sépare clarification et génération, tandis que la variante « one-step » les combine via un fine-tuning. Les auteurs évaluent SiriusBI sur les jeux SRD (requêtes single-round) et MRD (dialogues multi-round), ainsi qu'en production chez Tencent. Les résultats montrent une amélioration significative de l'exactitude des requêtes générées, notamment dans les contextes multi-tours.

TABLE I
PERFORMANCE NL2SQL DE SIRIUSBI SUR LE JEU SRD (UEX = USEFUL EXECUTION ACCURACY).

Modèle	**UEX (%)**
DIN-SQL (baseline)	30,06
MAC-SQL (baseline)	36,42
MRD-SQL (baseline)	61,85
SiriusBI (two-step)	79,19
SiriusBI (one-step)	83,24

Sur SRD, la version « one-step » atteint 83,24 % d'UEX, soit +21 points par rapport à MRD-SQL (61,85 %). Sur MRD, la même version obtient 62,50 % contre 28,13 % pour MRD-SQL fine-tuned. En production chez Tencent, la version « one-step fine-tuned » atteint 97 % de précision en finance, 89 %

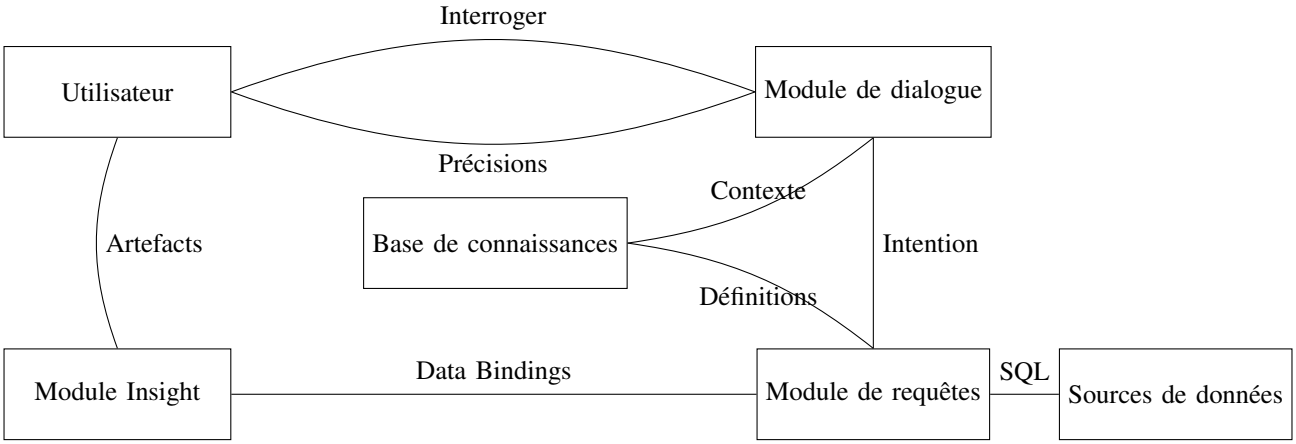

FIGURE 3. Architecture générale d'un système ChatBI. L'utilisateur saisit sa requête en langage naturel, que le Module de dialogue reformule et clarifie avant de la transmettre au Module de génération de requêtes. Ce dernier interagit avec la Base de connaissances pour construire des requêtes SQL conformes. Les Sources de données fournissent les résultats, que le Module Insight transforme en visualisations ou rapports visibles par l'utilisateur.

TABLE II
PERFORMANCE NL2SQL DE SIRIUSBI SUR LE JEU MRD (UEX = USEFUL EXECUTION ACCURACY).

Modèle	**UEX (%)**
DIN-SQL (baseline)	11,25
MAC-SQL (baseline)	15,63
MRD-SQL w/o SFT (baseline)	22,50
MRD-SQL (fine-tuned)	28,13
SiriusBI (two-step)	55,00
SiriusBI (one-step w/o SFT)	49,38
SiriusBI (one-step)	62,50

en publicité et 91 % en cloud [1], confirmant la robustesse de l'intégration du graphe et de MRD-Q.

B. AutoBIR (Busany et al., 2024)

Busany et al. (2024) présentent AutoBIR, un pipeline NL2SQL combinant génération automatique de requêtes et validation par tests unitaires. Leur architecture inclut un module de parsing du langage naturel, un générateur de prompts, un système de tests unitaires et une boucle de correction. Avant la validation finale, cinq experts métier (analystes BI et DBA seniors) ont examiné un échantillon de 200 requêtes pour en vérifier la pertinence fonctionnelle et la conformité aux conventions internes.

TABLE III
RÉSULTATS NL2SQL D'AUTOBIR SUR ADVENTUREWORKS2014.

Métrique	**Valeurs**
Exact Match	85,3 %
Execution Accuracy	88,7 %
Test-Case Pass Rate	92,4 %
Temps moyen de génération	1,8 s

Sur AdventureWorks2014, AutoBIR atteint 85,3 % d'Exact Match et 88,7 % d'Execution Accuracy, avec 92,4 % de réussite des tests unitaires. Le temps moyen de génération est de 1,8 s. La boucle de rétroaction auto-supervisée diminue de 60 % la consommation de tokens et augmente de 58 % la précision finale en corrigeant automatiquement les requêtes erronées.

L'ajout d'un module de génération de tests unitaires garantit la validité fonctionnelle avant exécution, réduisant ainsi drastiquement les erreurs. La revue experte a validé la qualité des prompts et la pertinence métier, confortant l'évolutivité du système à divers contextes.

Les performances NL2SQL de SiriusBI et d'AutoBIR confirment que les architectures combinant graphe de connaissances, dialogue multi-tours et boucles de rétroaction automatique génèrent des requêtes analytiques précises et fiables, tout en minimisant le coût en tokens et les erreurs d'exécution. Ces résultats ouvrent la voie à des systèmes BI plus agiles, où l'utilisateur obtient rapidement des réponses sans expertise technique approfondie.

VI. CONCLUSION

Dans cette revue de littérature, nous avons montré que l'intégration des LLMs dans les systèmes BI permet de simplifier considérablement la génération de requêtes analytiques, d'accélérer la prise de décision et d'élargir l'autonomie des utilisateurs métiers. Les études de cas SiriusBI [1] et AutoBIR [2] montrent qu'un pipeline structuré—du dialogue en langage naturel à la génération et à la validation automatique de code—peut atteindre une très bonne précision.

Au-delà de la simple agrégation de chiffres, un ChatBI pourrait également :

— proposer des scénarios de prévision (forecasting, budgétisation) en s'appuyant sur des modèles statistiques ou économétriques intégrés ;
— générer des analyses de corrélation ou de segmentation (k-means, clustering) pour identifier des comportements ou des segments de clients spécifiques ;

— interagir directement avec des modèles de machine learning externes afin d'intégrer des recommandations prédictives dans le processus décisionnel.

Ces extensions ouvrent la voie à une BI plus proactive et prédictive, où le LLM ne se contente plus de traduire une requête, mais devient un catalyseur d'analyses avancées. En implémentant ces cas d'usage, on pourra mesurer l'impact sur la capacité à anticiper les tendances, à optimiser les budgets et à personnaliser les recommandations pour chaque unité métier de Forester Inc.

En conclusion, exploiter les LLMs pour l'assistance BI ne se limite pas à automatiser la génération de requêtes : c'est une véritable opportunité pour transformer les systèmes d'information en plateformes d'analyse dynamique et prédictive.

RÉFÉRENCES

[1] J. Jiang, H. Xie, Y. Shen, Z. Zhang, M. Lei, Y. Zheng, Y. Fang, C. Li, D. Huang, W. Zhang, Y. Li, X. Yang, B. Cui, and P. Chen, "Siriusbi : Building end-to-end business intelligence enhanced by large language models," 2024. [Online]. Available : https ://arxiv.org/abs/2411.06102

[2] N. Busany, E. Hadar, H. Hadad, G. Rosenblum, Z. Maszlanka, O. Akhigbe, and D. Amyot, "Automating business intelligence requirements with generative ai and semantic search," 2024. [Online]. Available : https ://arxiv.org/abs/2412.07668

[3] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models : A survey," 2025. [Online]. Available : https ://arxiv.org/abs/2402.06196

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available : https ://arxiv.org/abs/1706.03762

[5] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.