Actes de CORIA-TALN 2026
Nantes, France, 29 juin au 3 juillet 2026

DQE : Génération de données diversifiées par agents pour le question-réponse sur tables

Raphaël Mouravieff[1,7] Tanguy Herserant[1,2,7] Arthur Satouf[4,5,6,7] Habiboulaye Amadou Boubacar[4] , Pablo Piantanida[5,6] Vincent Guigue[2] Sylvain Lamprier[3] Benjamin Piwowarski[1,7]
(1) Sorbonne Université, CNRS, Paris, France
(2) AgroParisTech - MIA, 22 place de l'Agronomie, 91120 Palaiseau, France
(3) Université Paris Cité, CNRS, LAMSADE, Paris, France
(4) Air liquide, Paris
(5) ILLS, MILA, Montreal, Canada
(6) Centralesupelec, CNRS, Universite Paris Saclay, Palaiseau, France
(7) ISIR, Paris, France
{tanguy.herserant, vincent.guigue}@agroparistech.fr

RÉSUMÉ

Les modèles de question-réponse sur tables (TableQA) présentent des échecs de généralisation face à des modifications simples de la structure des tables ou des questions posées. Nous soutenons que cela provient de la diversité limitée des jeux de données de pré-entraînement intermédiaire. Pour y remédier, nous introduisons un cadre multi-agents qui génère un ensemble diversifié et progressivement complexe de paires question–SQL. Notre cadre comprend trois agents spécialisés : un *agent curriculum*, un *agent traducteur* et un *agent diversificateur*. Le pré-entraînement d'un modèle basé sur BART sur le jeu de données résultant, DQE, améliore substantiellement la robustesse sur le benchmark ROBUT par rapport à des modèles de référence tels que TAPEX.

ABSTRACT

DQE : Diverse Data Generation via Agents for Table Question Answering

Table question answering (TableQA) models exhibit generalization failures when faced with simple modifications to either the table structure or the input question. We argue this stems from the limited diversity of intermediate pre-training datasets. To address this, we introduce a multi-agent framework that generates a diverse and progressively challenging set of question–SQL pairs. Our framework comprises three specialized agents : a curriculum agent, a translator agent, and a diversifier agent. Pre-training a BART-based model on the resulting DQE dataset substantially improves robustness on the ROBUT benchmark compared to strong baselines such as TAPEX.

MOTS-CLÉS : Question-réponse sur tables, génération de données, systèmes multi-agents, robustesse, SQL.

KEYWORDS: Table question answering, data generation, multi-agent systems, robustness, SQL.

Actes de la 21e Conférence en Recherche d'Information et Applications.
Volume 1 : articles scientifiques originaux, pages 94–106.
Cette œuvre est mise à disposition sous licence Attribution 4.0 International.

1 Motivation

Les LLM récents (Brown *et al.*, 2020) ont atteint des performances remarquables sur les benchmarks standards de TableQA (Dou *et al.*, 2023), mais présentent toujours une mauvaise généralisation face à des perturbations mineures des questions ou des structures de tables (Zhao *et al.*, 2023; Zhou *et al.*, 2024; Li *et al.*, 2025). Si une telle fragilité est attendue pour des modèles de fondation non explicitement entraînés pour le raisonnement tabulaire, elle reste surprenante pour des modèles ajustés finement (*fine-tuned*) qui subissent une adaptation au domaine (Liu *et al.*, 2021; Jiang *et al.*, 2022; Herzig *et al.*, 2020). La plupart des systèmes de TableQA adaptent des architectures textuelles (Devlin *et al.*, 2019; Lewis *et al.*, 2019) via un pré-entraînement intermédiaire sur des tâches liées aux tables avant un ajustement fin sur des jeux de données cibles (Pasupat & Liang, 2015; Chen *et al.*, 2019). Cependant, les corpus de pré-entraînement existants manquent souvent de diversité sémantique et syntaxique, étant largement basés sur des modèles (*templates*) ou annotés manuellement (Yu *et al.*, 2020; Shi *et al.*, 2022). Cette limitation peut introduire des décalages dans les distributions de formes logiques et de langage (Herzig & Berant, 2019), ou sur-spécialiser les modèles vers des motifs étroits et spécifiques au domaine. Par conséquent, les modèles tendent à sur-apprendre sur des formulations de questions spécifiques et échouent à traiter les variations linguistiques ou structurelles (Sui *et al.*, 2024). Nous soutenons que cette diversité limitée est une cause centrale de la fragilité en TableQA. Pour y remédier, nous introduisons un cadre multi-agents qui génère des paires question–SQL diversifiées et progressivement complexes.

Notre cadre de distillation de données (Radosavovic *et al.*, 2018) comprend un *agent curriculum* proposant de nouvelles questions, un *agent traducteur* générant des requêtes SQL exécutables validées par un moteur d'exécution, et un *agent diversificateur* améliorant la variété linguistique. Le pré-entraînement d'un modèle basé sur BART sur le jeu de données résultant, DQE, améliore substantiellement la robustesse sur le benchmark ROBUT, obtenant des gains importants par rapport à des modèles de référence tels que TAPEX. Ces résultats expérimentaux montrent que le pré-entraînement sur des données plus riches, générées automatiquement comme DQE, conduit à une meilleure généralisation dans les tâches de raisonnement tabulaire.

2 Méthodologie

Le cœur de notre approche est un pipeline basé sur des agents, conçu pour générer de manière autonome un jeu de données diversifié et de haute qualité pour le QA sur tables. Le système est composé de trois agents LLM principaux : Curriculum, Traducteur et Diversificateur, qui interagissent avec une bibliothèque centrale et le modèle en cours d'entraînement.

2.1 Outils

Notre cadre repose sur un ensemble d'outils dédiés qui permettent aux agents de valider, récupérer et exécuter des informations de manière cohérente lors de la génération du jeu de données. Ces outils servent d'interface fonctionnelle entre les agents et l'environnement de base de données sous-jacent.

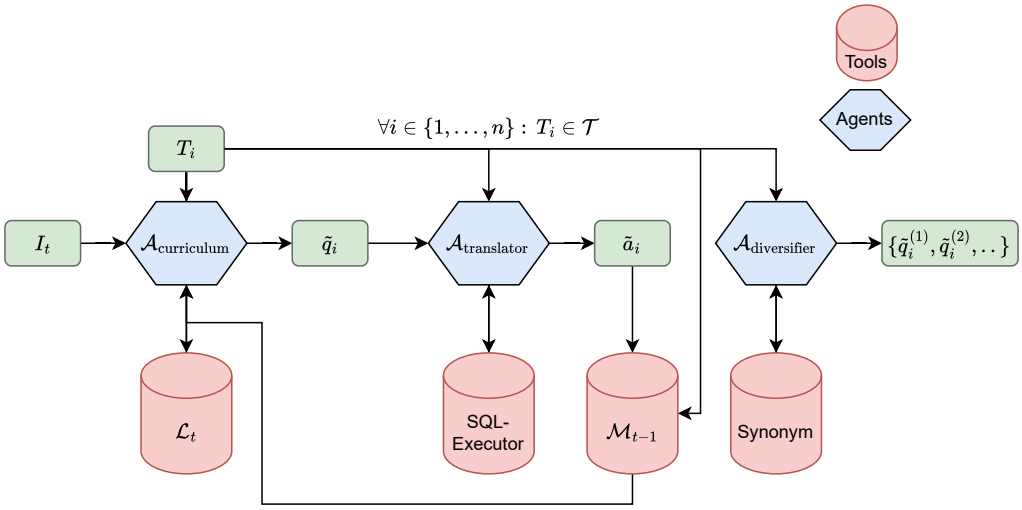

FIGURE 1 – Vue d'ensemble de notre pipeline de génération de données par agents. Une instruction I_t guide l'**agent curriculum** pour générer une nouvelle question $\tilde{q}_i$ pour une table donnée T_i, en fonction de l'état de la bibliothèque $\mathcal{L}_t$ et des performances du modèle courant $\mathcal{M}_{t-1}$. L'**agent traducteur** convertit $\tilde{q}_i$ en une requête SQL exécutable, produisant une réponse $\tilde{a}_i$. Enfin, l'**agent diversificateur** génère de multiples variations linguistiques de la question. Les nouvelles données sont utilisées pour ajuster finement l'itération suivante du modèle.

La bibliothèque. La bibliothèque, notée $\mathcal{L}$, sert de dépôt central de connaissances dans notre système. Elle stocke toutes les paires $(\text{question}, \text{SQL})$ générées avec succès tout au long du processus. Pour une récupération efficace et une vérification de nouveauté, chaque requête est représentée par un vecteur dense d'embedding calculé à l'aide de l'encodeur Alibaba-NLP (Zhang *et al.*, 2024). La similarité cosinus entre les embeddings est utilisée pour imposer deux contraintes sur toute requête nouvellement proposée $q^{(t+1)}$:

1. **Nouveauté :** La requête doit être suffisamment distincte de toutes les requêtes précédemment stockées : $\max_{i \in \{0..t\}} \cos(\phi_{\text{emb}}(q^{(t+1)}), \phi_{\text{emb}}(q^{(i)})) \leq \gamma_{max}$.
2. **Continuité :** Pour assurer un curriculum progressif, la requête doit rester proche en sens de la plus récente : $\cos(\phi_{\text{emb}}(q^{(t+1)}), \phi_{\text{emb}}(q^{(t)})) \geq \gamma_{min}$.

Ces contraintes fournissent un retour structuré E_t à l'agent curriculum, favorisant une exploration à la fois diversifiée et cohérente.

L'outil `retriever_tool`. L'outil `retriever_tool` est utilisé par l'agent curriculum pour accéder à la bibliothèque et évaluer si une question nouvellement générée est nouvelle. Il calcule la similarité cosinus entre l'embedding de la question candidate et ceux stockés dans $\mathcal{L}$, retournant un score de nouveauté. Si ce score dépasse le seuil γ_{max}, la candidate est rejetée et régénérée.

L'outil `execute_sql`. L'outil `execute_sql` permet à l'agent traducteur de valider ses requêtes SQL générées. Étant donné un schéma de table et une requête SQL, il exécute la requête via un moteur SQL léger $\mathcal{E}_{SQL}$ et retourne le résultat correspondant $\tilde{a}_i$. Cela garantit que chaque requête SQL est à la fois syntaxiquement valide et exécutable, produisant un triplet vérifiable $(\text{question}, \text{SQL}, \text{réponse})$.

2.2 Modèles de prompts

Chaque agent est guidé par un modèle de prompt dédié, définissant son rôle, ses entrées et ses sorties requises. Les prompts sont conçus pour assurer la cohérence, l'utilisation des outils et la vérifiabilité des sorties entre les agents. Plus précisément :

— **Agent curriculum (get_question_prompt) :** Génère une nouvelle question en langage naturel à partir d'un schéma de table donné et d'exemples récents stockés dans la bibliothèque. Le prompt impose la diversité, exige une référence explicite aux en-têtes de colonnes et requiert une validation via l'outil `retriever_tool`. (voir Section 2.1)
— **Agent traducteur (get_traductor_prompt) :** Convertit une question en langage naturel en une requête SQL exécutable, vérifiée à l'aide de l'outil `execute_sql`. Le prompt spécifie le schéma de la base de données, des exemples de tables, et demande une seule instruction SQL valide.
— **Agent diversificateur (get_extra_prompt_divers) :** Produit sept variantes paraphrasées suivant l'approche de Saparina & Lapata (2024) d'une question validée tout en conservant la requête SQL correspondante fixe. Le prompt liste explicitement les stratégies de paraphrase (simplification, remplacement par des synonymes, changement de perspective, etc.) et impose un format de sortie JSON.

Les modèles de prompts complets sont inclus en Annexe A à titre de référence et seront désignés comme instructions I_t dans la suite.

2.3 Agent curriculum

L'agent curriculum, $A_{curriculum}$, est responsable de piloter le processus d'exploration en proposant des questions nouvelles, pertinentes et progressivement complexes. À chaque étape t, pour une table donnée T_i, l'agent reçoit une instruction I_t et génère une question candidate $\tilde{q}_i$. Ce processus est guidé par deux boucles de rétroaction essentielles :

Vérification de nouveauté basée sur la bibliothèque : Pour éviter la redondance et encourager la diversité, la question générée est comparée aux questions existantes dans notre bibliothèque, $\mathcal{L}_t$. Nous utilisons un encodeur de phrases pré-entraîné « Alibaba-NLP » (Zhang *et al.*, 2024) pour calculer les embeddings de toutes les questions. Une nouvelle question $\tilde{q}_i$ est rejetée si sa similarité cosinus maximale avec toute question de la bibliothèque dépasse un seuil $\gamma_{max} = 0{,}9$. Cela garantit que chaque nouvelle question est suffisamment distincte de toutes les précédentes.

Vérification de difficulté basée sur le modèle : Pour créer un curriculum adapté aux faiblesses du modèle courant, nous utilisons le modèle de l'itération précédente, $\mathcal{M}_{t-1}$, comme discriminateur. Nous exécutons $\mathcal{M}_{t-1}(T_i, \tilde{q}_i)$ pour vérifier si le modèle peut déjà répondre correctement à la question proposée. La réponse de référence utilisée pour cette vérification correspond au résultat obtenu en exécutant la requête SQL validée via l'outil `execute_sql`, après traduction de $\tilde{q}_i$ en une requête SQL s_i par le module Traducteur décrit ci-dessous. Si la prédiction du modèle correspond à ce résultat, la question est rejetée car elle n'apporte pas de nouveau signal d'apprentissage. Ce mécanisme oriente l'agent vers la génération d'exemples difficiles pour le modèle courant, créant un curriculum dynamique et adaptatif.

2.4 Agent traducteur

Une fois que l'agent curriculum propose une question valide en langage naturel $\tilde{q}_i$, l'agent traducteur, $A_{translator}$, est chargé de la convertir en une requête SQL exécutable, $\tilde{s}_i$. Ce processus peut impliquer plusieurs tours de prompting itératif. La sortie SQL initiale de l'agent est passée à un moteur d'exécution SQL, $\mathcal{E}_{SQL}$. Si la requête échoue à s'exécuter ou produit un résultat incorrect, le retour d'erreur est fourni à l'agent, qui tente alors de corriger sa sortie. Cette paire vérifiée $(\tilde{s}_i, \tilde{a}_i)$, où $\tilde{a}_i = \mathcal{E}_{SQL}(\tilde{s}_i, T_i)$, assure la correction logique et syntaxique des données générées.

2.5 Agent diversificateur

Pour améliorer la diversité linguistique de notre jeu de données, l'agent diversificateur, $A_{diversifier}$, prend la question acceptée $\tilde{q}_i$ et génère de multiples paraphrases, par exemple $\{\tilde{q}_i^{(1)}, \tilde{q}_i^{(2)}, ...\}$. Ces variantes conservent l'intention sémantique originale (et donc la même requête SQL et la même réponse) mais utilisent des structures lexicales et syntaxiques différentes. Cette étape aide le modèle final à devenir robuste aux variations linguistiques superficielles, un point de défaillance courant des systèmes de QA sur tables.

2.6 Vue d'ensemble de l'algorithme

L'algorithme 1 résume le pipeline global de génération de données par agents décrit ci-dessus. Le processus génère itérativement de nouveaux triplets (question, SQL, réponse) tout en imposant des contraintes de nouveauté, d'exécutabilité et de difficulté, suivies d'une diversification linguistique.

3 Expériences

Nous menons des expériences pour valider notre hypothèse centrale : le pré-entraînement intermédiaire (Eisenschlos *et al.*, 2020) sur notre jeu de données généré de manière autonome (DQE) améliore la robustesse du modèle par rapport aux jeux de données existants.

3.1 Pipeline d'entraînement

Chaque agent de notre cadre est instancié comme un modèle Qwen2.5-14B (Hui *et al.*, 2024). Une seule étape du pipeline correspond au traitement d'une table de WikiTableQuestions à travers l'architecture multi-agents illustrée en Figure 1. Au cours de chaque étape, les agents collaborent pour générer une nouvelle requête SQL et sa question correspondante en langage naturel, le curriculum étant conçu pour augmenter progressivement la difficulté de la tâche. Chaque outil (par exemple, l'exécuteur SQL) peut être invoqué jusqu'à trois fois par les agents avant que le processus de génération ne se termine. Par exemple, l'agent responsable de la génération SQL peut vérifier au maximum trois fois si la requête produite s'exécute correctement avant que le processus ne s'arrête et passe à la table suivante. À l'étape suivante, le modèle mis à jour — ajusté finement sur les requêtes nouvellement générées — est réutilisé comme contrôleur pour guider le curriculum, orientant la génération vers

Algorithm 1 Génération de données par agents pour le QA sur tables

Require: Modèle initial $\mathcal{M}^0$, corpus de tables $D = \{T_1, \ldots, T_m\}$, bibliothèque $\mathcal{L}$

for itération globale $k = 0, 1, 2, \ldots$ **do**
  {Boucle externe : raffinement du modèle}
  for chaque table $T_i \in D$ **do**
    {Itération sur toutes les tables du corpus}
    Sélection du contexte : échantillonner quelques lignes de T_i pour fournir des exemples concrets.
    Préparation de l'instruction I_t **:** compiler le schéma de la table, les lignes échantillonnées et les entrées récentes de $\mathcal{L}$ pour guider $A_{\text{curriculum}}$.
    Agent curriculum : proposer une question candidate $\tilde{q}$ pour (T_i, I_t) et vérifier la nouveauté/continuité à l'aide de l'outil `retriever_tool`.
    if $\tilde{q}$ échoue à la contrainte de nouveauté **then**
      continuer {rejeter les questions redondantes}
    end if
    Agent traducteur : générer une requête SQL $\tilde{s}$ pour $\tilde{q}$ et la valider avec `execute_sql` ; soit $\tilde{a}$ la réponse retournée.
    if aucun SQL valide n'est trouvé **then**
      continuer
    end if
    if $\mathcal{M}^k(T_i, \tilde{q}) = \tilde{a}$ **then**
      continuer {écarter les éléments faciles}
    end if
    Agent diversificateur : produire des paraphrases $\{\tilde{q}^{(1)}, \ldots, \tilde{q}^{(7)}\}$ de $\tilde{q}$ en gardant $\tilde{s}$ fixe.
    Validation : ajouter $(\tilde{q}, \tilde{s}, \tilde{a})$ et ses paraphrases à $\mathcal{L}$.
  end for
  Entraînement du modèle : ajuster finement $\mathcal{M}^k$ sur les triplets accumulés $(\tilde{q}, \tilde{s}, \tilde{a})$ pour obtenir $\mathcal{M}^{k+1}$.
end for
return bibliothèque finale $\mathcal{L}$ et modèle $\mathcal{M}^{k+1}$

des exemples plus complexes et diversifiés. Pour des raisons d'efficacité computationnelle, nous rapportons les résultats obtenus après deux itérations de ce pipeline en raison de contraintes de temps et de ressources.

3.2 Configuration expérimentale

Benchmark d'évaluation : Nous évaluons notre approche sur le benchmark ROBUT (Zhao *et al.*, 2023), une référence standard pour mesurer la robustesse des modèles de QA sur tables. ROBUT introduit diverses perturbations adversariales sur les questions et les tables, telles que le remplacement de mots par des synonymes, la réorganisation des lignes/colonnes et l'ajout d'informations distractrices.

Métriques : La métrique officielle de ROBUT, la **précision robuste** (*Robust Accuracy*), mesure le pourcentage d'exemples correctement répondus à la fois avant et après perturbation. Cependant, elle n'est pas directement comparable entre les types de perturbation car le nombre d'exemples diffère avant et après perturbation. À la place, nous utilisons le **ratio de robustesse** (*Précision perturbée / Précision originale*), qui mesure la proportion de précision conservée après perturbation et permet une comparaison équitable entre les configurations.

Modèles et lignes de base. Nous utilisons l'architecture `BART-large` (Lewis *et al.*, 2019) comme

modèle de base, en suivant la même procédure d'entraînement et d'ajustement fin que TAPEX (Liu *et al.*, 2021) pour assurer une comparaison équitable. Nous comparons deux variantes : l'une pré-entraînée sur le jeu de données TAPEX, et l'autre pré-entraînée sur notre jeu de données DQE généré, en utilisant la même quantité de données d'entraînement que TAPEX.

3.3 Résultats de l'évaluation de robustesse

Comme le montre le Tableau 1, le modèle pré-entraîné sur notre jeu de données DQE surpasse systématiquement la ligne de base TAPEX sur presque tous les types de perturbation en termes de **ratio de robustesse**. Les gains les plus importants apparaissent pour les perturbations sémantiques telles que les *synonymes* d'en-tête (+8,4 p.p.), les *abréviations* d'en-tête (+5,8 p.p.) et l'*ajout* de contenu (+3,1 p.p.), indiquant que DQE aide le modèle à mieux préserver ses performances face à des modifications altérant le sens. Ces résultats suggèrent que le pré-entraînement sur DQE encourage une compréhension sémantique plus profonde plutôt qu'un apprentissage superficiel de motifs. La Figure 2 illustre ces améliorations. Notamment, les données de pré-entraînement de TAPEX sont construites autour de la distribution de WikiTableQuestions et reposent sur des modèles SQL artisanaux issus de SQUALL (Shi *et al.*, 2020), tandis que notre processus de génération piloté par agents explore une gamme plus large et plus diversifiée de paires question–SQL.

Type de perturbation	**TAPEX (%)** Ratio de robustesse	**DQE (%)** Ratio de robustesse	**Variation** (DQE – TAPEX)
abréviation (en-tête)	92,2	**98,0**	+5,8
synonyme (en-tête)	91,1	**99,4**	+8,4
mot (question)	89,4	**93,1**	+3,7
phrase (question)	90,8	**91,2**	+0,5
masqué (contenu)	**90,6**	87,7	-2,9
colonne (contenu)	88,9	**90,6**	+1,6
ajout (contenu)	84,3	**87,4**	+3,1
ligne (contenu)	79,1	**80,9**	+1,8
combiné	80,1	**83,1**	+3,0
extension (contenu)	**73,5**	73,3	-0,2

TABLE 1 – Résultats du ratio de robustesse. DQE conserve une proportion plus élevée de sa précision originale que TAPEX sur la plupart des types de perturbation, en particulier ceux impliquant des variations sémantiques.

3.4 Analyse du jeu de données

Pour comprendre pourquoi notre jeu de données produit des modèles plus robustes, nous avons analysé ses propriétés internes par rapport à TAPEX.

Diversité syntaxique SQL. Nous utilisons la bibliothèque SQLGlot [1] pour analyser chaque requête SQL et extraire son arbre syntaxique, ce qui nous permet d'analyser la distribution des types de nœuds (c'est-à-dire les mots-clés et opérateurs SQL). Comme le montre la Figure 3, notre jeu de données

1. https://github.com/tobymao/sqlglot

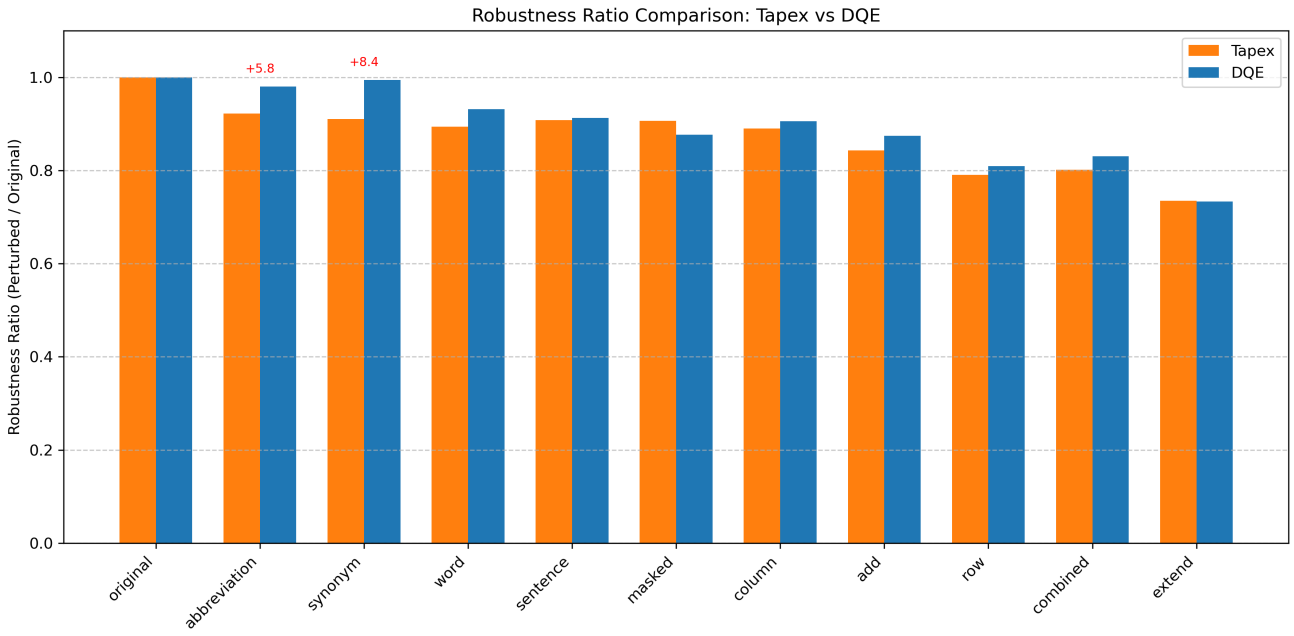

FIGURE 2 – Comparaison de robustesse sur ROBUT. DQE surpasse TAPEX sur 8/11 types de perturbation (par exemple, synonyme +8,4 p.p., abréviation +5,8 p.p., ajout +3,1 p.p.), est à égalité sur l'original, et TAPEX est légèrement meilleur sur masqué (-2,9 p.p.) et extension (-0,2 p.p.).

Métrique	**DQE**	**TAPEX**	**Différence**
Nombre de requêtes	10 000	10 000	0
Types de nœuds uniques	69	38	31
Nombre total de nœuds	185 550	77 260	108 290
Types de nœuds rares ($\geq$10)	12	6	6
Types de nœuds communs	35	35	0
Types de nœuds exclusifs	34	3	31

TABLE 2 – Comparaison des statistiques de graphes SQL entre notre jeu de données généré et TAPEX.

DQE couvre une gamme substantiellement plus large de fonctionnalités SQL. DQE inclut 69 types de nœuds uniques contre seulement 38 dans TAPEX, avec 34 types de nœuds exclusifs à notre jeu de données. Le Tableau 2 résume quantitativement ces différences, démontrant la capacité du cadre d'agents à explorer et générer des structures SQL plus complexes et diversifiées.

Couverture de l'espace sémantique : Nous avons projeté 10 000 requêtes SQL de chaque jeu de données dans un espace 2D en utilisant UMAP sur leurs embeddings. La Figure 4 confirme visuellement la plus grande diversité de DQE. Alors que les échantillons TAPEX sont concentrés dans un noyau dense, les échantillons DQE couvrent une zone significativement plus large, indiquant que notre méthode d'exploration par agents découvre avec succès des régions plus diversifiées de l'espace sémantique.

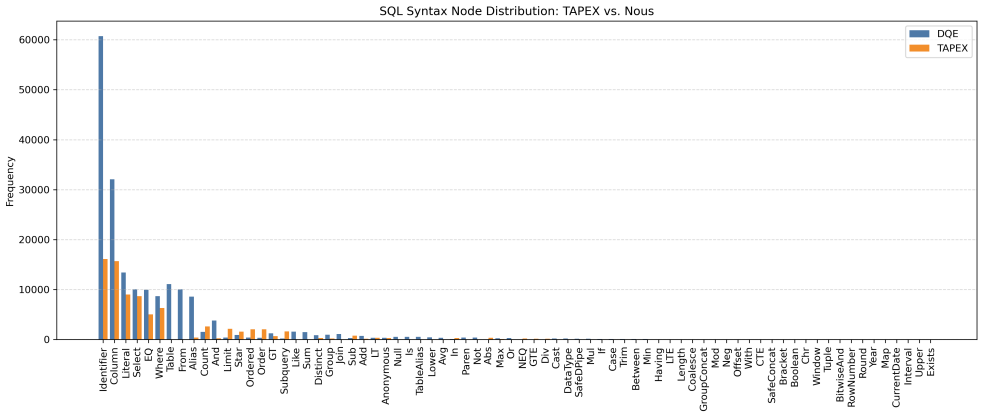

FIGURE 3 – Distribution des nœuds syntaxiques SQL. DQE présente une plus grande variété de structures et de mots-clés SQL par rapport à TAPEX, mettant en évidence un raisonnement compositionnel plus riche.

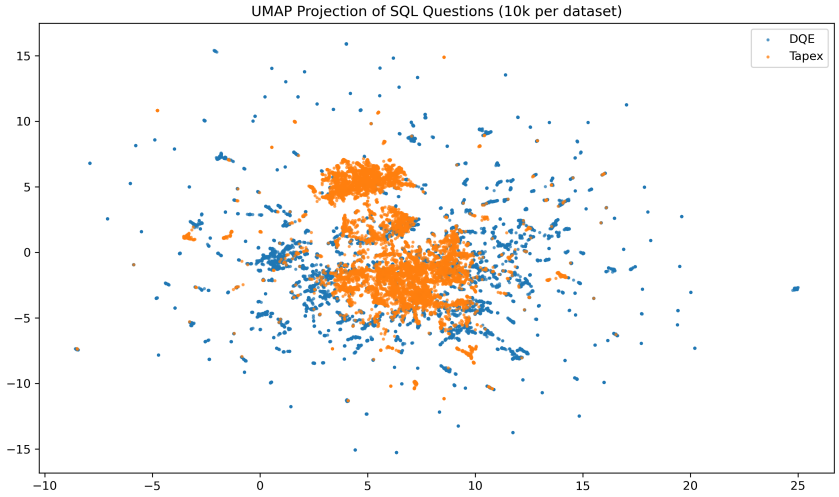

FIGURE 4 – Projection UMAP des embeddings de requêtes SQL. DQE (bleu) couvre un espace sémantique beaucoup plus large que TAPEX (orange), indiquant une plus grande diversité.

4 Conclusion

Nous avons présenté un nouveau cadre basé sur des agents qui génère de manière autonome un jeu de données diversifié et de haute qualité pour le QA sur tables. En employant une approche guidée par un curriculum avec des agents distincts pour la proposition de questions, la traduction et la diversification, notre système explore l'espace SQL et du langage naturel de manière plus efficace que les méthodes traditionnelles de création statique de jeux de données. Nos expériences montrent que le pré-entraînement sur nos données générées conduit à des modèles avec une robustesse substantiellement améliorée face aux perturbations adversariales.

5 Limites

Bien que notre cadre basé sur des agents montre un fort potentiel pour améliorer la robustesse et la diversité des données, quelques limites subsistent.

Curriculum adaptatif. Le curriculum repose actuellement sur des seuils de similarité heuristiques pour imposer la nouveauté. Des travaux futurs pourraient intégrer des stratégies adaptatives ou basées sur l'apprentissage par renforcement qui ajusteraient dynamiquement la difficulté des questions en fonction des performances du modèle.

Efficacité computationnelle. La nature séquentielle du pipeline multi-agents augmente le coût computationnel en raison des invocations répétées du LLM et de la vérification SQL. Le développement de variantes plus légères ou parallélisées pourrait conserver la diversité tout en améliorant la scalabilité.

Portée de l'évaluation. Notre évaluation se concentre sur le benchmark ROBUT. Étendre les expériences à des jeux de données text-to-SQL supplémentaires (par exemple, WikiSQL, Spider) fournirait une évaluation plus complète de la généralisation inter-domaines.

Références

BROWN T., MANN B., RYDER N., SUBBIAH M., KAPLAN J. D., DHARIWAL P., NEELAKANTAN A., SHYAM P., SASTRY G., ASKELL A. *et al.* (2020). Language models are few-shot learners. *Advances in neural information processing systems*, **33**, 1877–1901.

CHEN W., WANG H., CHEN J., ZHANG Y., WANG H., LI S., ZHOU X. & WANG W. Y. (2019). Tabfact : A large-scale dataset for table-based fact verification. *arXiv preprint arXiv :1909.02164*.

DEVLIN J., CHANG M.-W., LEE K. & TOUTANOVA K. (2019). Bert : Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics : human language technologies, volume 1 (long and short papers)*, p. 4171–4186.

DOU L., GAO Y., PAN M., WANG D., CHE W., ZHAN D. & LOU J.-G. (2023). Multispider : towards benchmarking multilingual text-to-sql semantic parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, p. 12745–12753.

EISENSCHLOS J. M., KRICHENE S. & MÜLLER T. (2020). Understanding tables with intermediate pre-training. *arXiv preprint arXiv :2010.00571*.

HERZIG J. & BERANT J. (2019). Don't paraphrase, detect ! rapid and effective data collection for semantic parsing. *arXiv preprint arXiv :1908.09940*.

HERZIG J., NOWAK P. K., MÜLLER T., PICCINNO F. & EISENSCHLOS J. M. (2020). Tapas : Weakly supervised table parsing via pre-training. *arXiv preprint arXiv :2004.02349*.

HUI B., YANG J., CUI Z., YANG J., LIU D., ZHANG L., LIU T., ZHANG J., YU B., LU K. *et al.* (2024). Qwen2. 5-coder technical report. *arXiv preprint arXiv :2409.12186*.

JIANG Z., MAO Y., HE P., NEUBIG G. & CHEN W. (2022). Omnitab : Pretraining with natural and synthetic data for few-shot table-based question answering. *arXiv preprint arXiv :2207.03637*.

LEWIS M., LIU Y., GOYAL N., GHAZVININEJAD M., MOHAMED A., LEVY O., STOYANOV V. & ZETTLEMOYER L. (2019). Bart : Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv :1910.13461*.

LI C., LIU X., SONG Z., CHI C., ZHAO C., YANG J., WANG Z., YANG K., SHI B., WANG X., DENG C. & FENG J. (2025). Treb : A comprehensive benchmark for evaluating table reasoning capabilities of large language models. *arXiv preprint arXiv :2506.18421*.

LIU Q., CHEN B., GUO J., ZIYADI M., LIN Z., CHEN W. & LOU J.-G. (2021). Tapex : Table pre-training via learning a neural sql executor. *arXiv preprint arXiv :2107.07653*.

PASUPAT P. & LIANG P. (2015). Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv :1508.00305*.

RADOSAVOVIC I., DOLLÁR P., GIRSHICK R., GKIOXARI G. & HE K. (2018). Data distillation : Towards omni-supervised learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, p. 4119–4128.

SAPARINA I. & LAPATA M. (2024). Improving generalization in semantic parsing by increasing natural language variation. *arXiv preprint arXiv :2402.08666*.

SHI P., NG P., NAN F., ZHU H., WANG J., JIANG J., LI A. H., CHAKRAVARTI R., WEIDNER D., XIANG B. *et al.* (2022). Generation-focused table-based intermediate pre-training for free-form question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, p. 11312–11320.

SHI T., ZHAO C., BOYD-GRABER J., DAUMÉ III H. & LEE L. (2020). On the potential of lexico-logical alignments for semantic parsing to sql queries. *arXiv preprint arXiv :2010.11246*.

SUI Y., ZHOU M., ZHOU M., HAN S. & ZHANG D. (2024). Table meets llm : Can large language models understand structured table data ? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, p. 645–654.

YU T., WU C.-S., LIN X. V., WANG B., TAN Y. C., YANG X., RADEV D., SOCHER R. & XIONG C. (2020). Grappa : Grammar-augmented pre-training for table semantic parsing. *arXiv preprint arXiv :2009.13845*.

ZHANG X., ZHANG Y., LONG D., XIE W., DAI Z., TANG J., LIN H., YANG B., XIE P., HUANG F. *et al.* (2024). mgte : Generalized long-context text representation and reranking models for multilingual text retrieval. *arXiv preprint arXiv :2407.19669*.

ZHAO Y., ZHAO C., NAN L., QI Z., ZHANG W., TANG X., MI B. & RADEV D. (2023). Robut : A systematic study of table qa robustness against human-annotated adversarial perturbations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, p. 6064–6081.

ZHOU W., MESGAR M., ADEL H. & FRIEDRICH A. (2024). FREB-TQA : A fine-grained robustness evaluation benchmark for table question answering. In K. DUH, H. GOMEZ & S. BETHARD, Éds., *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies (Volume 1 : Long Papers)*, p. 2479–2497, Mexico City, Mexico : Association for Computational Linguistics. DOI : 10.18653/v1/2024.naacl-long.137.

A Modèles de prompts

Cette annexe liste le texte complet des trois modèles de prompts utilisés dans notre cadre de distillation de données.

A.1 Prompt de l'agent curriculum (`get_question_prompt`)

```
You are an SQL **question generator** in natural
language for a table with the following structure.

DATABASE INFOS: {{ table_info }}
DATABASE SCHEMA: {{ table_schema }}
Clean Table random samples: {{ table_clean }}
Dirty Table random samples: {{ table_dirty }}

{% if vector_store %}
NOTE: The library already contains
{{ vector_store_size }} questions.
Recent questions in the library (up to 3 most recent):
{{ example_questions }}
Your NEW question should be similar in complexity
but *not* a duplicate. Try to query different tables
or relationships to avoid exact overlap.
{% endif %}

Using the Dirty Table random samples, generate a
clear and specific **natural language question** that
can be answered using the database structure above.

IMPORTANT CRITERIA:
1. Specific enough to be translated into SQL.
2. Must have an answer in the database.
3. Don't code the table.
4. Write in plain, clear natural language.
5. Use **the dirty table header** to reference columns.
6. Use "retriever_tool" to validate the question.
7. Return: "final_answer(retriever_tool(question))"

Return **only the question**.
```

A.2 Prompt de l'agent traducteur (`get_traductor_prompt`)

```
You are an SQL **question generator** in natural
language for a database with the following structure.

DATABASE INFOS: {{ table_info }}
DATABASE SCHEMA: {{ table_schema }}
Dirty Table random samples: {{ table_dirty }}
```

```
Clean Table random samples: {{ table_clean }}
Natural Language Question: {{ question }}

INSTRUCTION:
Write a single valid SQL query that correctly answers
the Natural Language Question using the database
structure provided above. JOIN tables if needed.
You must use the "execute_sql" tool to check if the
SQL is valid.
Finally return the valid SQL:
"final_answer(execute_sql(sql_query))"

RETURN ONLY the SQL query without explanations,
commentary, or markdown formatting.
```

A.3 Prompt de l'agent diversificateur (`get_extra_prompt_divers`)

```
You are a **question paraphrasing expert**.

Original question: {{ question }}
Its Corresponding SQL query: {{ sql_question }}
Table: {{ table_dirty }}

Your task is to create 7 different variants of the
original question using the following techniques:
1. BASIC SIMPLIFICATION
2. SIMPLIFICATION BY HIDING DETAILS
3. USING SYNONYMS
4. SEMANTIC SUBSTITUTIONS
5. COMPLETE REFORMULATION
6. SIMPLIFICATION WITH MORE DIRECT LANGUAGE
7. PARAPHRASE WITH CHANGE OF PERSPECTIVE

For each original SQL question, generate all 7
variants without changing the SQL query.
Use the 'get_synonym' tool when needed.

Return a JSON array with the following format:
"final_answer([
    {"question": "Variant 1"},
    ...
    {"question": "Variant 7"}
])"
```