

Rapport DM LO17

Michael FERNANDEZ, Rayan BARREDDINE, Valentin RONSSERAY, Jiawen WU

Table des matières

| | |
|--|---|
| Table des matières..... | 1 |
| I) Motivations du choix de l'univers Pokémon comme sujet pour notre RAG..... | 1 |
| II) Développement du RAG..... | 1 |
| A. Collecte des données..... | 1 |
| B. Implémentation des modèles de RAG..... | 3 |
| III) Streamlit..... | 4 |
| IV) Evaluation..... | 5 |
| V) Conclusion..... | 8 |

I) Motivations du choix de l'univers Pokémon comme sujet pour notre RAG

Dans le cadre de ce projet, notre objectif initial était de développer un système de RAG (*Retrieval-Augmented Generation*) sur le thème du sport. Cependant, au fil de nos recherches, nous avons constaté une limitation importante : il était difficile de trouver des sources fiables, structurées et suffisamment riches en contenu pour alimenter efficacement notre base documentaire. La diversité des disciplines sportives et la rareté de certaines informations détaillées aurait rendu le projet difficilement exploitable.

Face à ces contraintes, nous avons décidé de réorienter notre sujet vers un univers à la fois riche, bien documenté et apprécié du grand public : l'univers Pokémon. Ce thème offre un grand nombre de ressources en ligne (sites spécialisés, encyclopédies collaboratives, etc.), ce qui en fait un terrain idéal pour la mise en place d'un RAG. Nous nous sommes notamment penchés sur le site PokéPédia qui couvre la très grande majorité des informations relatives à l'univers Pokémon.

II) Développement du RAG

A. Collecte des données

Pour développer notre système de RAG (*Retrieval-Augmented Generation*) sur l'univers Pokémon, nous avons choisi de nous appuyer sur deux sources de données: PokéPédia et PokéAPI. PokéPédia est une encyclopédie en ligne extrêmement complète et régulièrement mise à jour par la communauté. Cette base de données est particulièrement pertinente pour le développement d'un RAG car elle comporte beaucoup de textes, ce qui est important pour la vectorisation des données. Afin de constituer notre base de connaissances, nous avons

utilisé l'API du site dans le but d'extraire automatiquement les liens internes du site et de récupérer les pages les plus pertinentes.

Grâce à cette méthode, nous avons pu collecter un grand nombre de pages couvrant une variété de sujets : descriptions de Pokémon, types, objets, lieux, personnages, etc. Cependant, cette approche a rapidement montré ses limites. En effet, l'extraction automatique a généré un volume de données beaucoup trop important, incluant des milliers de liens et de contenus, ce qui a dépassé notre capacité de traitement, à la fois en termes de mémoire et d'efficacité du moteur de recherche embarqué dans le RAG.

Pour remédier à ce problème, nous avons donc décidé de sélectionner **uniquement les Pokémon de la première génération ainsi que leurs extensions introduites dans la quatrième génération** (formes alternatives, Méga-Évolutions, etc.)

Nous avons dû effectuer un tri sélectif dans les données récupérées. Nous avons notamment fait le choix de supprimer certaines catégories d'informations moins essentielles, comme les pages détaillant les capacités (attaques, statistiques associées, effets spécifiques, etc.), qui représentaient une part significative du corpus total. Cette réduction ciblée nous a permis d'alléger la base documentaire tout en conservant les éléments les plus utiles pour répondre aux requêtes courantes sur l'univers Pokémon.

Ce filtrage a été une étape essentielle pour garantir un bon équilibre entre richesse de l'information et performance du système.

Notre deuxième source de données est PokéAPI, une base donnée accessible via une API développeur donnant des données précises et structurées sur les Pokémon. Nous récoltons alors toutes ces informations :

- Nom, forme (ex : Méga-Évolution), identifiant, taille, poids, expérience de base, etc.
- Capacités (abilities), incluant les talents standards et talents cachés.
- Types (ex : Roche / Vol), utilisés pour filtrer ou regrouper les Pokémon.
- Statistiques de base : points de vie (HP), attaque, défense, vitesse, etc., ainsi que les valeurs d'effort (EVs).
- Espèce et taxonomie : nom d'espèce, couleur, habitat, taux de capture, bonheur de base, rythme de croissance, classification (ex : Pokémon Fossile).
- Genres et reproduction : taux de répartition mâle/femelle, nombre de cycles d'éclosion.
- Descriptions multilingues issues de différents jeux Pokémon (flavor text par version, en français, anglais et japonais).

Un exemple typique de fichier JSON utilisé est reporté en Annexe (Figure 2 - Structure type JSON utilisée (*stats_index.json*))

L'avantage de cette source de données est que l'on aura des informations précises sur les Pokémon. En revanche, l'inconvénient est qu'il n'y a très peu de texte dans ces données, rendant difficile la vectorisation des données par le modèle d'embedding et par conséquent la pertinence des réponses retournées.

Pour améliorer nos données, nous avons en plus de cela créer des index sur la couleur, l'habitat, le statut ou encore le type à partir des données de PokéAPI. (Se reporter à l'annexe Figure 3)

Après avoir scrappé l'ensemble des données, nous les avons formaté grâce à quelques scripts Pythons afin d'optimiser le stockage des données. Au total, nous disposons alors de 211 documents PokeAPI + 154 documents Poképédia + les index.

B. Implémentation des modèles de RAG

Prochaine étape, implémenter les modèles de RAG à l'aide de nos données. Nous avons alors suivi l'ensemble des instructions transmises dans le Notebook. Premièrement, nous émettons une clé API Google AI Studio, puis nous la plaçons dans le fichier d'environnement `.env`.

Nous avons porté beaucoup d'attention sur la qualité de l'implémentation du RAG dans notre programme, notamment grâce à de la programmation orientée objet. Nous avons alors créé une classe RAGSystem.

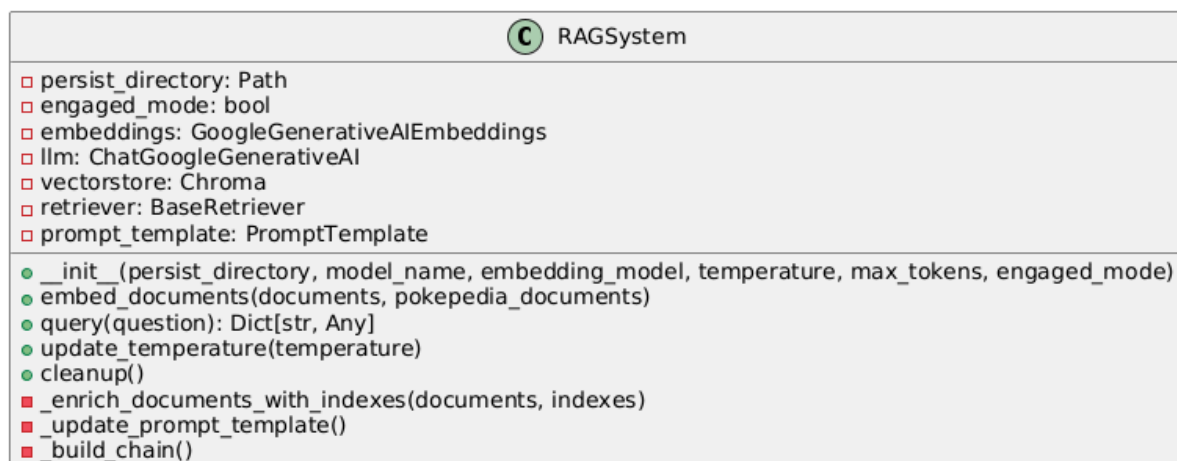


Figure 1 - UML Classe RAGSystem

Nous initialisons la RAG comme ceci. Comme conseillé dans le notebook, nous utilisons le LLM `"gemini-2.0-flash"` (`"gemini-2.5-flash"` ne marche pas) et le model d'embedding `"models/embedding-001"`. Nous implémentons aussi la température et `engaged_mode`, qui nous serviront à implémenter des fonctionnalités supplémentaires.

```
class RAGSystem:
    def __init__(
        self,
        persist_directory: str = "./chroma_db",
        model_name: str = "gemini-2.0-flash",
        embedding_model: str = "models/embedding-001",
        temperature: float = 0.0,
        max_tokens: int = 256,
        engaged_mode: bool = False,
    )
```

Figure 2 - Définition de la classe RAGSystem (*rag_core.py*)

La classe met en place toute l'architecture d'un moteur RAG spécialisé Pokémon. Dès son instantiation, elle crée un dossier Chroma temporaire, puis choisit le mode de réponse "normal" pour des réponses courtes ou "engagé" pour des explications riches en adaptant le prompt à l'un ou l'autre. Cela nous a permis de voir la différence entre les réponses selon les prompts et le nombre de documents utilisés dans le contexte.

Pour pouvoir raisonner sur les données, nous utilisons la méthode `embed_documents()` qui vectorise puis indexe un corpus combinant Poképédia, PokeAPI et index. Elle commence par charger les articles PokéPédia et les JSON de PokéAPI, filtre les structures compliquées, puis alimente la BDD ChromaDB. La base vectorielle ainsi constituée est exposée via un retriever, configuré pour renvoyer 2 documents en mode normal ou 4 en mode engagé, afin de doser la quantité de contexte fourni au LLM.

Lorsqu'on appelle `query()`, la fonction vérifie la présence du retriever, rappelle la température courante du modèle et lance la recherche sémantique : les documents les plus proches de la question sont récupérés. Ils sont ensuite formatés, injectés dans un prompt spécial Pokémon et envoyés au LLM qui produit la réponse finale. Le résultat retourné inclut la réponse et le contexte exact utilisé.

Enfin, quelques utilitaires complètent l'ensemble : `update_temperature()` permet de régler la créativité du modèle, `_format_docs()` permet d'assembler proprement le contexte, `cleanup()` et le destructeur `__del__()` veillent à supprimer le dossier Chroma temporaire pour éviter toute fuite de ressources. Ainsi, de l'ingestion des données jusqu'à la génération de la réponse, le flux reste entièrement maîtrisé et nettoyé.

Ainsi, nous respectons l'ensemble des principes de la programmation SOLID.

III) Streamlit

Pour rendre notre système RAG interactif et accessible, nous avons développé une interface web avec Streamlit, en centralisant la logique dans un fichier principal nommé `app.py`. Ce

fichier expose le système via une interface ergonomique, pensée pour l'expérimentation et la démonstration locale.

Dès le démarrage, les données sont chargées et intégrées grâce à la fonction `create_pokemon_documents`, qui prépare les documents à indexer. Grâce aux sessions Streamlit, l'état des embeddings est conservé, ce qui évite un rechargement complet à chaque rafraîchissement de la page et offre une expérience utilisateur optimale.

L'interface se compose de plusieurs éléments interactifs :

- Un champ de texte principal où l'utilisateur peut saisir sa question librement.
- [Bonus] Des métriques indiquant le score de fidélité entre le contexte extrait et la réponse générée.
- [Bonus] Dès lors que la probabilité d'hallucination dépasse 30 %, le chatbot déclare ne pas connaître la réponse.
- Un espace de saisie d'une réponse de référence, permettant de comparer manuellement la sortie du système.
- Les passages du corpus cités.

La barre latérale offre un ensemble de paramètres et d'informations utiles :

- [Bonus] Le réglage de la température du modèle, influençant la créativité de la réponse.
- Une liste d'exemples de questions pour guider l'utilisateur dans ses requêtes.
- Des statistiques sur le corpus intégré, telles que le nombre de pokémons ou la couverture des types de Pokémon.

Enfin, une section dédiée à l'évaluation permet à l'utilisateur de sauvegarder ses propres réponses de référence afin d'enrichir la base de comparaison et affiner le suivi des hallucinations générées par le système.

Nous pouvons voir un aperçu de la page en Annexe (Figure 4 Aperçu du Streamlit)

IV) Evaluation

L'évaluation du RAG Pokémon repose sur deux familles de métriques : d'une part les indicateurs RAGAS (faithfulness, answer relevancy, context precision, context recall) et, d'autre part, des mesures internes (similarité textuelle, chevauchement de mots-clés, précision factuelle). Les douze questions du jeu de tests sont traitées par lots afin de respecter la limite de 15 requêtes/minute de l'API Gemini. Après chaque lot, les résultats

intermédiaires sont enregistrés puis le système attend 1 minute avant de poursuivre. Voici les résultats des évaluations:

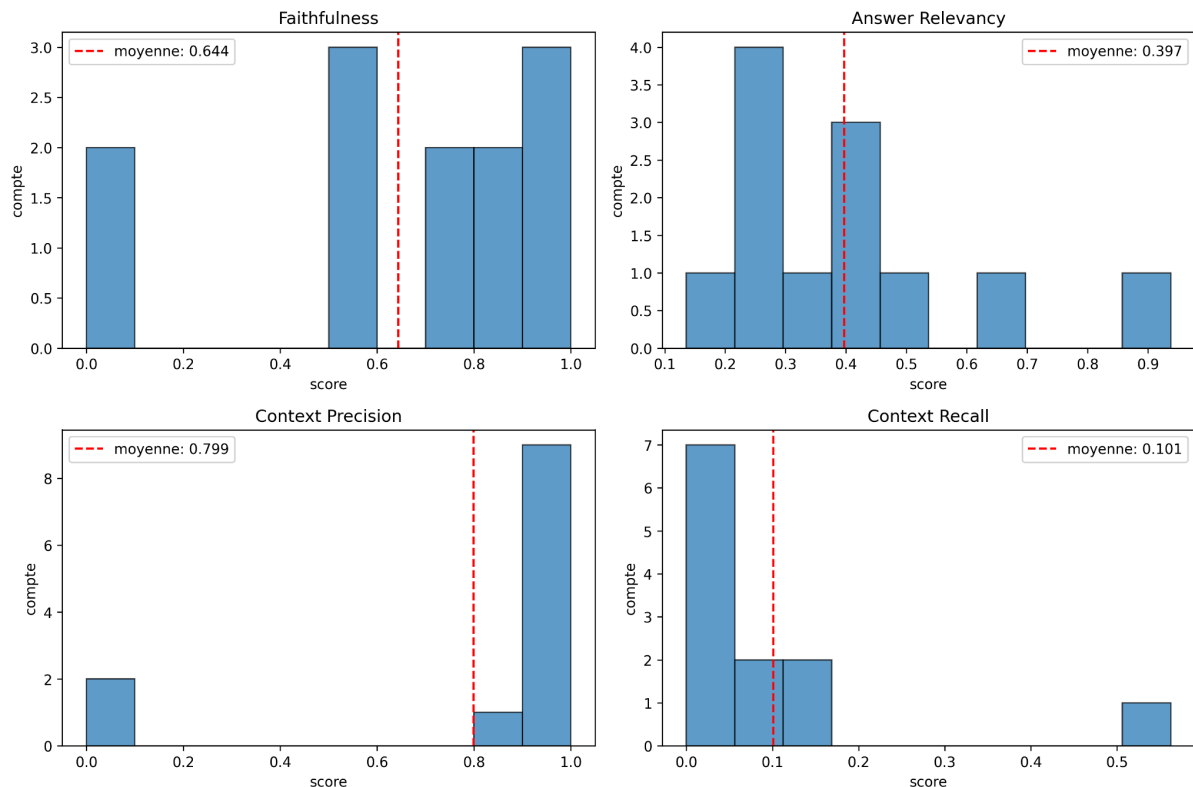


Figure 2 - Résultats des évaluations de notre système

| Statistique | Faithfulness | Answer Relevancy | Context Precision | Context Recall |
|-------------|--------------|------------------|-------------------|----------------|
| Mean | 0.644 | 0.397 | 0.799 | 0.101 |
| Std | 0.353 | 0.215 | 0.376 | 0.153 |
| Min | 0.000 | 0.136 | 0.000 | 0.000 |
| Max | 1.000 | 0.938 | 1.000 | 0.562 |
| Median | 0.739 | 0.356 | 0.952 | 0.050 |

En moyenne, le système obtient 0,64 de faithfulness et 0,80 de context précision, ce qui montre qu'il exploite avec justesse les passages qu'il a récupérés. En revanche, la couverture globale (context recall à 0,10) et la pertinence perçue (answer relevancy à 0,40) restent faibles car les bonnes informations sont trouvées mais pas toujours utilisées de façon exhaustive ni parfaitement alignées sur la question.

Les résultats varient fortement selon la catégorie : les requêtes sur l'évolution atteignent 1,00 de faithfulness et de précision, tandis que les comparaisons qui nécessitent des données manquantes dans le corpus tombent à zéro. Les questions de statistiques restent

solides sur la véracité factuelle mais pèchent en recall, faute d'exploiter toutes les informations disponibles.

Le système RAG Pokémon exploite très précisément le contexte : 75 % des réponses dépassent 0,9. Il montre une fiabilité parfaite sur les questions d'évolution et reste factuellement solide pour les statistiques individuelles. Cependant, la pertinence globale reste faible, avec 83 % des réponses sous 0,5, et le rappel du contexte demeure limité, surtout pour les listes complètes. Des lacunes documentaires persistent concernant les Pokémon légendaires, mythiques et les questions de comparaison.

Nous avons alors testé l'évaluation avec le mode engagé pour voir la différence entre les deux modes :

| Statistique | Faithfulness | Answer Relevancy | Context Precision | Context Recall |
|-------------|--------------|------------------|-------------------|----------------|
| Mean | 0.686 | 0.437 | 0.857 | 0.032 |
| Std | 0.315 | 0.126 | 0.111 | 0.023 |
| Min | 0.000 | 0.248 | 0.667 | 0.007 |
| Max | 1.000 | 0.619 | 1.000 | 0.092 |
| Median | 0.701 | 0.450 | 0.842 | 0.027 |

Nous constatons que en mode engagé, les moyennes montent pour la faithfulness (0,69 vs 0,64), la answer relevancy (0,44 vs 0,40) et surtout la context précision (0,86 vs 0,80), mais le context recall baisse (0,03 vs 0,10), signe que le système cite moins d'informations différentes.

Les écarts-types chutent presque partout, ce qui indique des performances plus régulières et prévisibles dans le mode engagé. En contrepartie, le meilleur score de pertinence atteint 0,62 contre 0,94 en mode normal et la médiane de précision recule, confirmant qu'un contexte plus restreint rend les réponses globalement plus cohérentes mais limite les pointes d'excellence et la couverture totale des sources.

Pour progresser, il faut enrichir le corpus avec les fiches manquantes et affiner les métadonnées pour une meilleure catégorisation. Un prompt plus direct encouragerait des réponses plus complètes et mieux alignées sur la question. Enfin, l'ajout d'une recherche hybride mêlant vecteurs et mots-clés améliorerait la récupération d'informations encore absentes.

V) Conclusion

Ce projet a permis de démontrer comment un système RAG peut efficacement combiner des données issues de la PokéAPI avec des contenus textuels plus narratifs issus de Poképédia pour offrir un assistant riche, fiable et accessible autour de l'univers Pokémon. L'architecture du projet facilite la mise à jour du corpus et l'ajout de nouveaux documents.

L'interface utilisateur développée avec Streamlit joue un rôle central dans l'expérience globale. Elle permet à chacun d'explorer l'univers Pokémon en posant des questions naturelles, tout en bénéficiant d'indicateurs de confiance et de possibilités d'évaluation intégrées. Son design est assez simple et permet à tout utilisateur de comprendre rapidement comment l'utiliser et obtenir les informations souhaitées.

Le projet permet d'avoir une transparence des sources et une traçabilité des données : tous les documents utilisés pour générer les réponses sont conservés sous forme brute dans des formats ouverts et sont retournés lors des réponses. Cela permet non seulement de vérifier l'origine des réponses. Un utilisateur peut facilement enrichir la précision ou tester de nouveaux cas sans modifier la structure existante.

Le mode engagé gagne un peu en fait, pertinence et précision, mais perd encore en recall en citant moins d'informations. Les deux modes restent fiables pour les évolutions et statistiques, mais peinent toujours sur les listes, comparaisons et Pokémon légendaires. La chute des écarts-types en mode engagé traduit une performance plus stable, au prix d'un plafond de pertinence inférieur au mode normal. Enrichir le corpus, affiner les métadonnées, muscler le prompt et combiner recherche vectorielle + mots-clés aideront à combler les lacunes et élargir la couverture du contexte.

Enfin, ce projet illustre parfaitement le potentiel des systèmes RAG appliqués à des bases de connaissances complexes : il est à la fois suffisamment général pour servir de modèle reproductible, et suffisamment spécifique pour répondre à des besoins concrets relatifs à l'univers Pokémon. Que ce soit pour développer un assistant conversationnel, créer un outil pédagogique ou étudier le comportement des modèles de langage, ce RAG Pokémon constitue une bonne base pour d'autres projets divers et variés.

Contribution : La contribution a été équitablement répartie entre les quatre membres du groupe, chacun ayant pris en charge environ 25 % des tâches liées au développement, ou à la rédaction du rapport.

Annexe

Pikachu

Page

Discussion

Lire

Modifier

Voir l'historique

Outils

Arbok

0024

Pikachu


0025

Raichu

0026

N° 0025

Pikachu



Artwork de Pikachu pour *Pokémon Rouge Feu* et *Vert Feuille*.

Nom

Pikachu

Nom japonais

ピカチュウ

Nom anglais

Pikachu

Numéros de Pokédex régionaux

| Kanto | Johto ^{ORAS} | Johto ^{FGSS} | Hoenn ^{RSE} |
|-----------------------|-----------------------|---------------------------|----------------------|
| 025 | 022 | 022 | 156 |
| Hoenn ^{ROSA} | Sinnoh | Kalos ^(Centre) | Alola ^{SL} |
| 163 | 104 | 036 | 025 |
| Alola ^{USUL} | Galar | Isolarmure | Hisui |
| 032 | 194 | 085 | 056 |

Pikachu (anglais : **Pikachu** ; japonais : **ピカチュウ** *Pikachu*^[1]) est un **Pokémon** Souris de **type Électrik** apparu dès la **première génération**. En tant que partenaire de **Sacha**, héros du **dessin animé** tiré du jeu, il est le plus célèbre des Pokémon et la mascotte officielle de la licence.

À propos du Pokémon

[modifier]

Physionomie et attitudes

[modifier]

Pikachu est un petit Pokémon joufflu qui ressemble à un rongeur, au corps recouvert d'un pelage jaune avec deux bandes horizontales brunes dans le dos. Il a une petite bouche, de longues oreilles pointues aux extrémités noires et des yeux noir et blanc. Il a sur chacune de ses joues un marquage rouge qui est en réalité une poche contenant de l'électricité. Il a cinq petits doigts au bout de chacun de ses membres antérieurs, tandis que ses pattes postérieures ont trois petits orteils. Il a de la fourrure brune à la base de sa queue en forme d'éclair. Bien que catégorisé comme quadrupède, il est capable de se tenir et de se déplacer sur ses pattes postérieures.

Pikachu n'a pas été conçu au départ comme un rongeur ou une souris mais comme une simple créature ronde. Toutefois, ses poches d'électricité sont inspirées des abajoues que certains rongeurs, dont l'écureuil, utilisent pour stocker de la nourriture et il a finalement été décidé de le présenter comme une souris. Ses oreilles en pointe noire et le motif de son dos ne sont inspirés d'aucun animal et la forme en éclair de sa queue a été choisie pour expliciter son type Électrik^[2].

Pikachu a deux formes alternatives : le **Pikachu Cosplayeur**, apparu et disponible uniquement dans *Pokémon Rubis Oméga* et *Saphir Alpha*, est toujours de sexe féminin et a un marquage noir en forme de cœur au bout de la queue. Il peut être déguisé en l'un des cinq déguisements disponibles, chacun correspondant à une **condition de concours Pokémon** ; le **Pikachu à Casquette**, apparu comme **Pokémon événementiel** dans la **septième génération**, est toujours de sexe masculin et a sept variantes, chacune portant l'une des casquettes portées par **Sacha Ketchum** dans le **dessin animé**, et dont la **capacité Z** signature est **Giga-Tonnerre** ; le **Pikachu partenaire**, qui peut être de n'importe quel sexe, mais qui a de meilleures statistiques qu'un Pikachu ordinaire et qui peut apprendre des capacités exclusives : **Pika-Sprint**, **Pika-Piqué**, **Pika-Splash** et **Pika-Fracas**.

Figure 1 - Page Poképédia Pikachu

```

{
  "abilities": [
    {
      "ability": {
        "name": "static"
      },
      "is_hidden": false,
      "slot": 1
    },
    {
      "ability": {
        "name": "lightning-rod"
      },
      "is_hidden": true,
      "slot": 3
    }
  ],
  "base_experience": 112,

```

Figure 2 - Index sur le statut des pokémons (*statuts_index.json*)

10/14

```
{
  "legendary": [
    "articuno-galar",
    "articuno",
    "mewtwo-mega-x",
    "mewtwo-mega-y",
    "mewtwo",
    "moltres-galar",
    "moltres",
    "zapdos-galar",
    "zapdos"
  ],
  "mythical": [
    "mew"
  ],
  "baby": []
}
```

Figure 3 - Index sur le statut des pokémons (*statuts_index.json*)

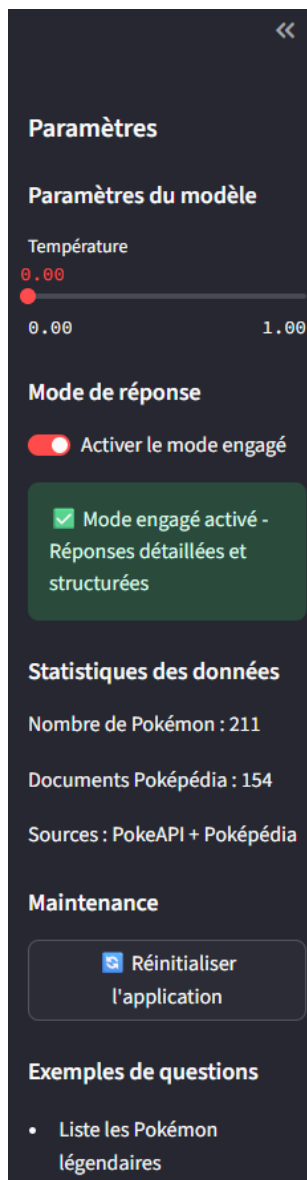


Figure 4 - Aperçu du Streamlit

⚡ Pokédex IA - Système de Questions-Réponses

Deploy

Cette application utilise un système RAG (Retrieval-Augmented Generation) pour répondre à vos questions sur les Pokémon. Le système utilise le modèle Gemini de Google pour la génération et ChromaDB pour le stockage et la récupération des informations.

Les données proviennent de deux sources principales :

- **PokeAPI** : Informations détaillées sur chaque Pokémon (statistiques, types, capacités, descriptions officielles)
- **Poképédia** : Contenu enrichi en français avec descriptions détaillées, biologie, comportement, habitat, mythologie et faits divers

Le système utilise une recherche vectorielle avancée avec :

- Métadonnées enrichies incluant les informations d'index (types, statuts, habitats, couleurs)
- Intégration automatique des données Poképédia pour des réponses plus riches et détaillées
- Recherche sémantique pour comprendre le contexte et l'intention des questions

Posez votre question

Entrez votre question:

Liste les Pokémon légendaires

Recherche sémantique (vecteurs)

Réponse

Voici quelques faits intéressants sur Arcanin :

- **Apparence et inspiration** : Arcanin est un Pokémon quadrupède et canin avec une fourrure orange marquée par des stries noires. Il est inspiré des komainu, des statues de lions gardiens chinois. (Poképédia)
- **Personnalité et capacités** : Arcanin est décrit comme fier et loyal, capable de courir 10 000 kilomètres en 24 heures. Son pouvoir provient d'une flamme ardente dans son corps. (Poképédia)
- **Habitat** : Arcanin vit dans les prairies, mais peut aussi vivre dans les volcans actifs. (Poképédia)
- **Variantes** : Depuis *Légendes Pokémon : Arceus*, il existe une variante appelée Arcanin de Hisui. (Poképédia)
- **Type** : Arcanin est un Pokémon de type Feu. (Poképédia)
- **Talents** : Il possède les talents Intimidation, Torche et Cœur Noble.
- **Statistiques de base** : Ses statistiques sont : PV 90, Attaque 110, Défense 80, Attaque Spéciale 100, Défense Spéciale 80, Vitesse 95.
- **Évolution** : Arcanin est l'évolution de Caninos lorsqu'il est exposé à une Pierre Feu. (Poképédia)
- **Description** : "Son port altier et son attitude fière ont depuis longtemps conquis le cœur des hommes."
- **Arcanin de Hisui** : Il est de type Feu et Roche, avec les talents Intimidation, Torche et Tête de Roc. Ses statistiques de base sont : PV 95, Attaque 115, Défense 80, Attaque Spéciale 95, Défense Spéciale 80, Vitesse 90.

Indicateurs de Confiance

Probabilité d'Hallucination

46.8%

Recouvrement du Contexte

53.2%

Confiance Globale

⚠ Attention : Cette réponse pourrait contenir des informations incorrectes ou inventées.

Voir le Contexte Récupéré

Contexte 1 📄 (pokepedia):

Informations poképédia sur arcanin:

Arcanin (anglais : Arcanine ; japonais : ウィンディ Windie [1]) est un Pokémon de type Feu de la première génération . Il est parfois à tort considéré comme légendaire , du fait du nom de sa famille pouvant prêter à confusion.

Arcanin est un Pokémon quadrupède et canin avec une fourrure orange marquée par des stries noires, qui font penser à des déchirures. Il a des oreilles en forme de diamant avec du beige à l'intérieur, un nez noir et rond et deux dents pointues qui dépassent de sa mâchoire supérieure. Sa tête, sa gueule et sa poitrine sont recouvertes par une fourrure beige et hirsute, sauf autour des oreilles et des yeux. De longues touffes de fourrure poussent derrière ses genoux et autour de ses chevilles. Son ventre est noir et il a une grosse queue hirsute et beige. Chaque patte a trois orteils et un coussinet rose.