Présentation Projet de Recherche Sémantique

par Nathan Cerisara

Plan de cette présentation

1) Introduction du sujet
2) Démos (WebApp puis Bot dans Rainbow)
3) Explications du fonctionnement / Architecture du projet
4) Conclusion
5) Questions?

Introduction

- Recherche Syntaxique dans Rainbow
- Limitations: Fautes de frappes, pas les bons mots
  - Ex:
    - “La réunion de 10h” ↛ “La réu de 10h”
    - “J’ai du mal à travailler” ↛ “J’arrive pas à bosser”
- Solution
  - -> Recherche sémantique

Démo - WebApp

Démo - Intégration dans Rainbow

Grosses limitations, là ce n’est qu’un petit bot.

De meilleures performances + bien meilleure interface / interaction si jamais c’est bien développé dans Rainbow directement.

Explication du fonctionnement - Plan

1) Formalisation d’une instance Rainbow
2) Structure de moteurs / Exemple d’un moteur de recherche
3) Optimisations / Stockage de cache de données
4) Architecture du Serveur
5) Intégration à Rainbow / Bot de recherche

1) Formalisation d’une instance Rainbow

Bubble

- id
- name
- members_ids
- messages_ids

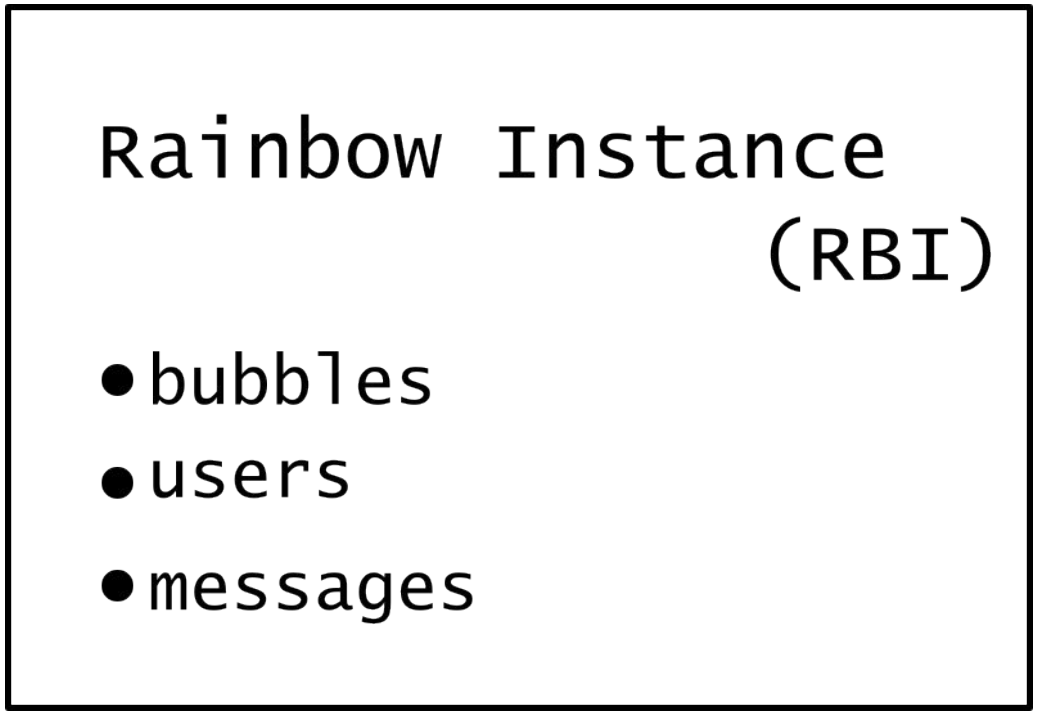
User

- id
- name
- bubbles_ids
- messages_ids

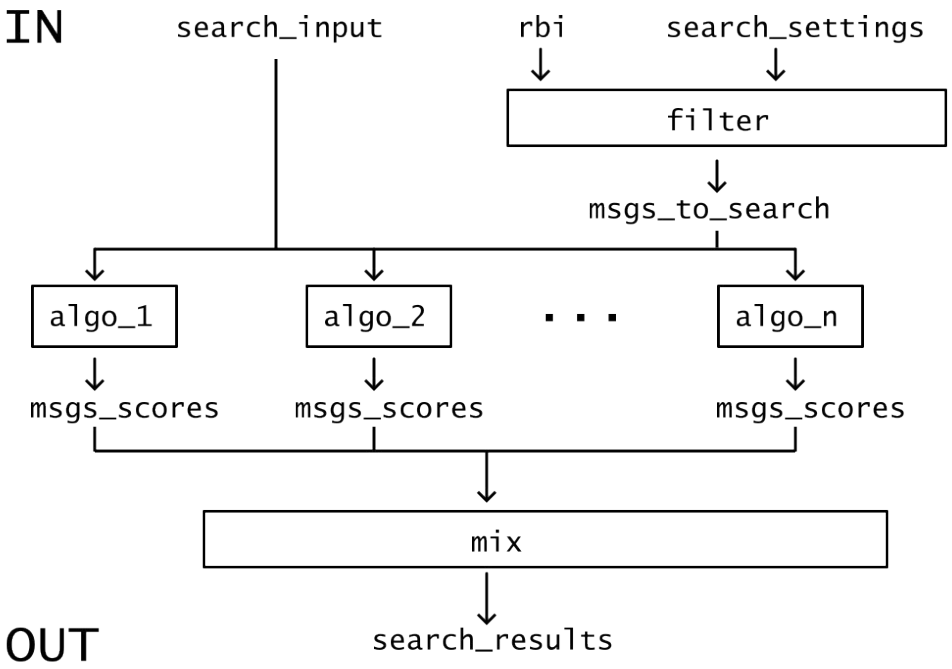
Message

- id
- content
- author_id
- author_name
- date
- bubble_id
- answered_msg_id

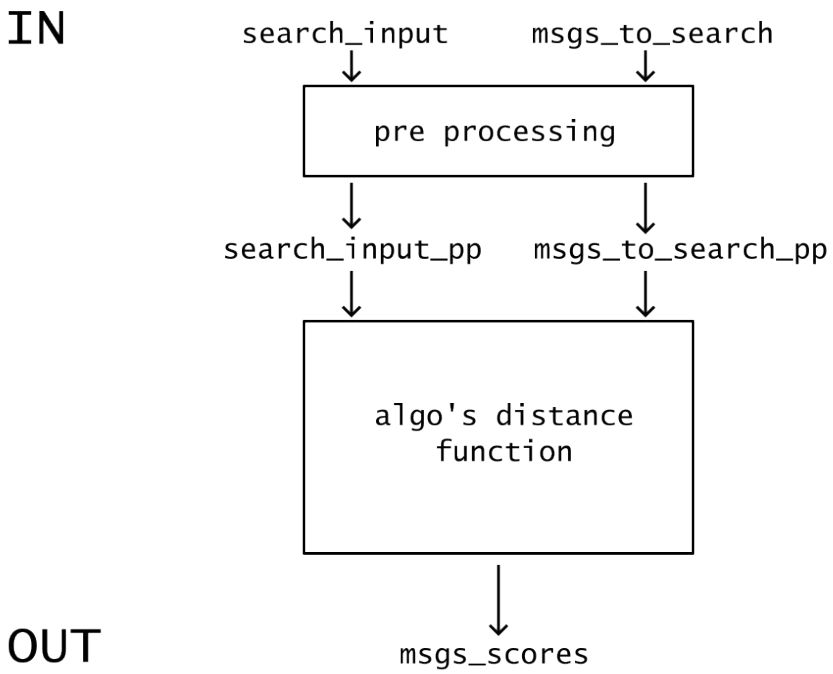
1) Formalisation d’une instance Rainbow



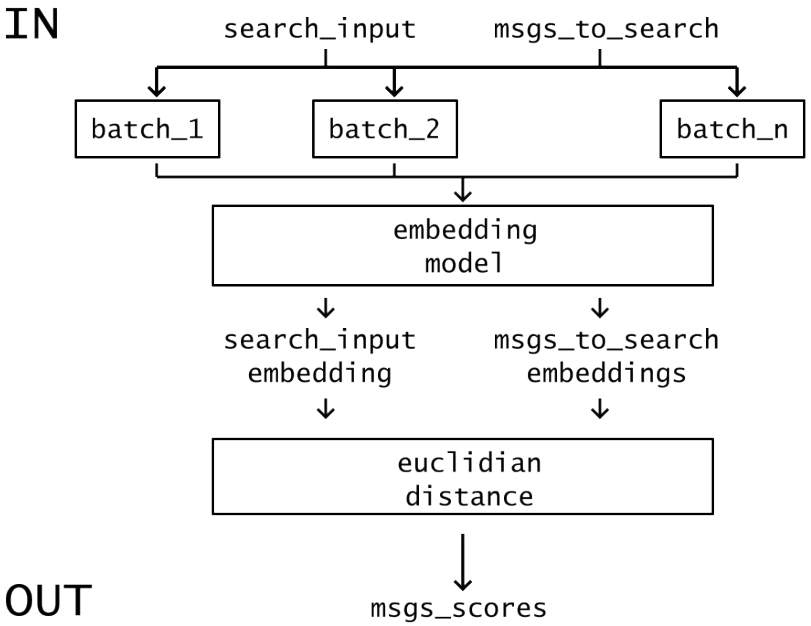
2) Structure de moteurs / Exemple d’un moteur de recherche

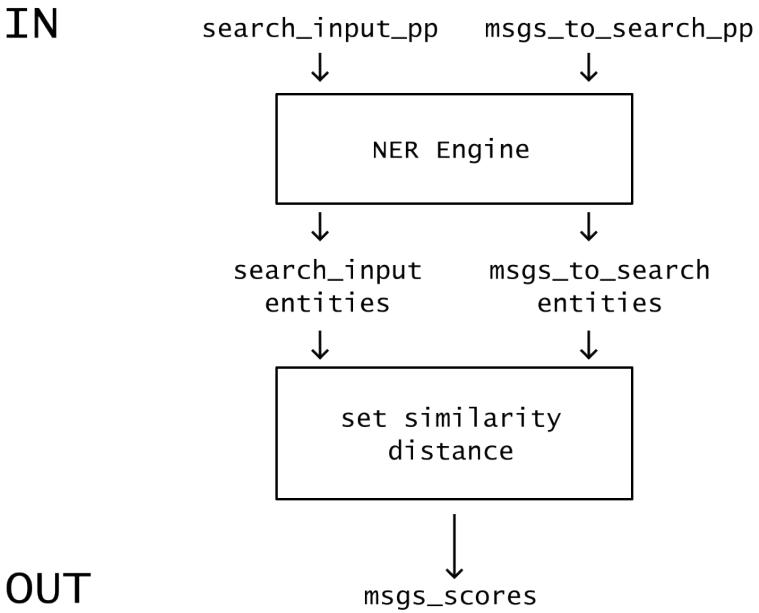


2) Structure de moteurs / Exemple d’un moteur de recherche



2) Structure de moteurs / Exemple d’un moteur de recherche





2) Structure de moteurs / Exemple d’un moteur de recherche

Modèle d’embedding:

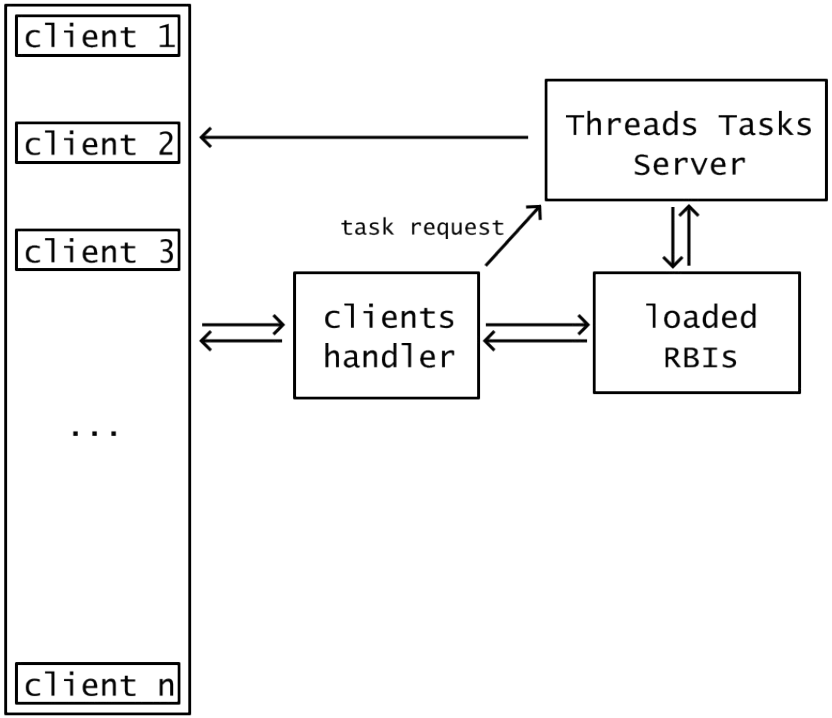
- Architecture Transformers
- AllMiniLM-l6-v2 (depuis HuggingFace), licence Apache 2.0
  - 22 millions de paramètres
  - Architecture de type Bert
  - 88 Mo sur le disque dur
  - Tourne en local sur ce PC

Plus de détails dans les questions pour ceux qui veulent.

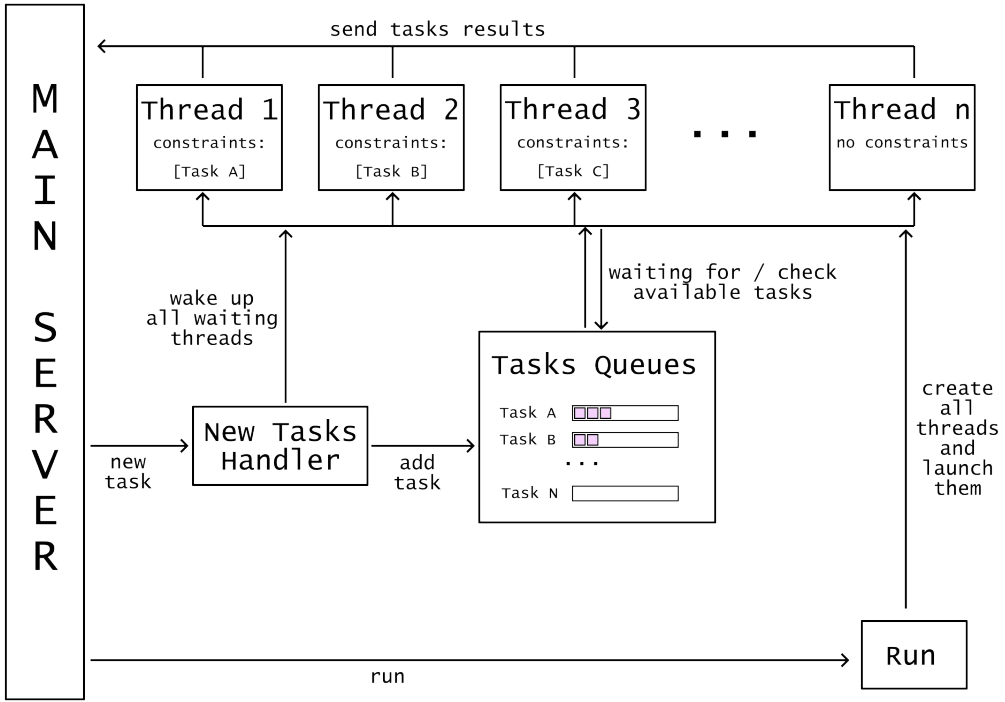
3) Optimisations / Stockage de cache de données

- Utilisation de poids de modèles quantizés (4 bits, gain de ~1 sec / messages)
- Cache d'embeddings (gain: ~ 0.2 sec / messages)
- Traduction (meilleures performances, voir benchmarks)
- Cache de traduction (gain: ~ 5 sec / messages)
- Avec modèles d'embeddings meilleurs en Français
  - On peut se passer de traduction
- Stratégie actuelle: tout pré-calculer une seule fois à l'initialisation du serveur + mises à jours rapides à la réception de messages, tout côté serveur.

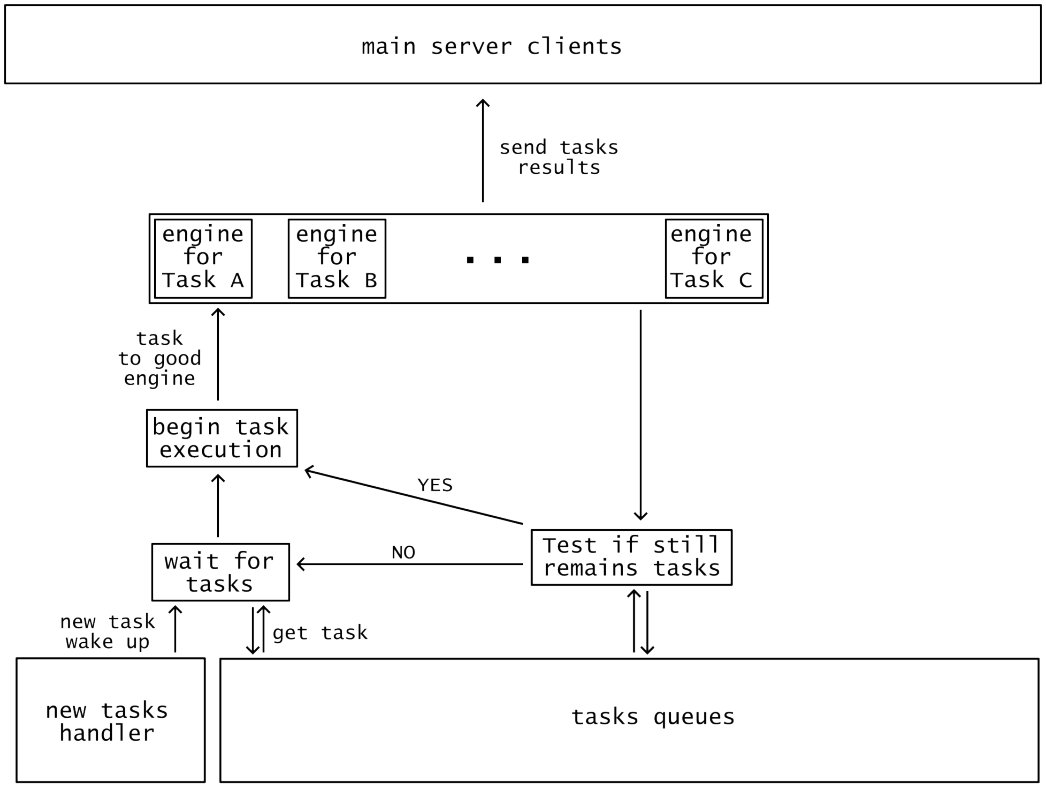
4) Architecture du Serveur



4) Architecture du Serveur



4) Architecture du Serveur



5) Intégration à Rainbow / Bot de recherche

- Sandbox Rainbow
- Utilisation SDK C#
- Connexion socket entre script C# et serveur Python
- Limitations de la SDK
    - On n’a donc qu’un petit bot qui fonctionne par messages
        - Si intégré à Rainbow, ce sera directement dans l’interface de la recherche

4) Conclusion

- Meilleure Recherche
- Architecture modulable -> flexibilité et améliorations
- Avec meilleurs modèle d’embeddings pour le Français
  - pas besoin de traduction, donc plus de problèmes de cache de traduction
- Il reste quand mêmes quelques limites sur “l’intelligence” du modèle.
- Tout tourne en local sur le laptop Thinkpad
  - + rapide avec meilleurs CPU / GPU
- Il faudra quand même bien étudier la mise à l’échelle, pour voir quelle puissance serveur sera requise, ou bien déléguer certaines tâches côté client?

Questions ?