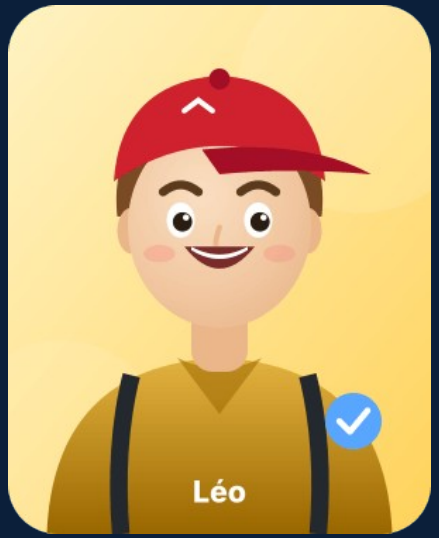
HEF-160 · MISSION 04 / 10

RAG et recherche de connaissances

Au lieu de demander au modèle ce qu'il croit savoir, on va chercher la bonne information dans vos documents et on lui demande de répondre à partir de ce qu'on lui montre.



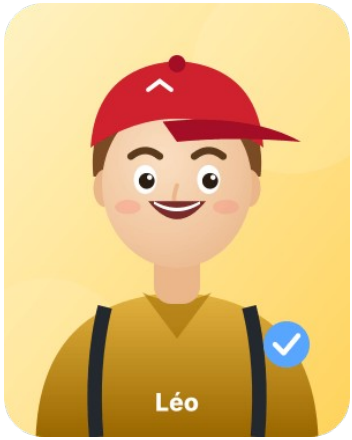


Badge : Bibliothécaire de l'IA

110 min · avancé

L'histoire

HashBox veut un assistant qui répond aux questions des membres sur le règlement, les remboursements et les certificats. Le premier essai interroge directement un modèle : il répond avec aplomb des règles qui n'existent pas (« remboursement sous 60 jours » alors que HashBox rembourse sous 14 jours). Maya propose une autre approche : « Quand une question arrive, on cherche d'abord les passages pertinents de nos propres documents, on les donne au modèle, et on exige qu'il réponde *uniquement* à partir d'eux, en citant la source. Et s'il n'y a rien de pertinent, il dit qu'il ne sait pas. »



« C'est le principe du RAG, ajoute Sam : Retrieval-Augmented Generation. Ça ne supprime pas toutes les erreurs, mais ça remplace "ce que le modèle croit se rappeler" par "ce qu'on peut lui montrer et vérifier". Tout se joue dans la qualité de la recherche — que l'on peut mesurer. »

L'histoire

L'histoire (suite)

Tu apprends à découper des documents en morceaux, à indexer et rechercher (TF-IDF, BM25, similarité cosinus), à fusionner plusieurs classements, à mesurer la qualité de la recherche (rappel@k, MRR), à produire des réponses avec citations, et à refuser de répondre quand l'information manque.

Ta mission

À la fin, je serai capable de…

1. décrire le pipeline RAG : découpage, indexation, récupération, assemblage, génération, citation
2. implémenter une recherche lexicale (TF-IDF, BM25) et une similarité par vecteurs
3. mesurer la qualité de la récupération (rappel@k, MRR) et l'effet de la taille des morceaux
4. fusionner des classements (fusion de rangs réciproques) et reclasser
5. répondre de façon extractive avec citations et refuser quand la preuve est insuffisante

Défi d'échauffement

Défi d'échauffement

Un étudiant passe un examen à livre ouvert avec 300 pages de cours. Stratégie A : tout lire pendant l'examen. Stratégie B : utiliser l'index et la table des matières pour trouver les 3 pages utiles, puis répondre en citant la page. Que faut-il pour que B fonctionne ? Que se passe-t-il si l'index est mauvais ? Que doit faire l'étudiant si les pages trouvées ne contiennent pas la réponse ?

Le concept

Le pipeline RAG



Description du schéma : les documents sont découpés puis indexés. Pour une question, on recherche les meilleurs morceaux, on les assemble avec des consignes, le modèle génère une réponse citant ses sources, ou refuse si la preuve manque.

Pourquoi le RAG

Problème	Apport du RAG
Connaissances absentes ou datées	on récupère des documents à jour

Le concept

Pourquoi le RAG (suite)

Problème	Apport du RAG
Données privées	elles ne sont pas dans le modèle, on les fournit à la demande
Hallucinations	la réponse s'appuie sur des passages vérifiables
Traçabilité	on cite les sources
Coût de ré-entraînement	on met à jour l'index, pas le modèle

Découper (chunking)

Choix	Effet
Petits morceaux (100-300 mots)	récupération précise, mais contexte perdu
Grands morceaux (500-1000 mots)	contexte riche, mais bruit et coût
Chevauchement (10-20 %)	évite de couper une réponse en deux

Le concept

Découper (chunking) (suite)

Choix	Effet
Découpage par structure (titres, sections)	préserve le sens
Métadonnées (titre, date, source)	filtrage, citation

La taille idéale dépend des documents et des questions : **se mesure** (rappel@k).

Lexicale (mots)

- **TF-IDF** : un mot compte d'autant plus qu'il est **fréquent dans le morceau** (TF) et **rare dans le corpus** (IDF = log(N / df)).
- **BM25** : amélioration de TF-IDF avec saturation de la fréquence et normalisation de la longueur ; référence solide.

Le concept

Lexicale (mots) (suite)

```
BM25(q, d) = Σ_t IDF(t) · tf·(k1+1) / (tf + k1·(1 − b + b·|d|/avgdl))
```

Forte pour les termes exacts (noms, codes), faible pour les synonymes.

Vectorielle (sémantique)

Chaque texte devient un **vecteur** (plongement, *embedding*) ; les textes de sens proche sont proches. On compare par **similarité cosinus**. Forte pour les paraphrases, plus faible pour les termes exacts rares. (Ici, on illustre le principe avec des vecteurs de mots simples ; les vrais plongements viennent d'un modèle.)

Hybride et reclassement

Le concept

Hybride et reclassement (suite)

Combiner lexical et sémantique, puis éventuellement **reclasser** (*rerank*) les meilleurs candidats avec un modèle plus précis. La **fusion de rangs réciproques** (RRF) combine des classements sans calibrer les scores :

```
score(d) = Σ_classements 1 / (k + rang(d))      (k ≈ 60)
```

Mesurer la récupération

On construit un **jeu d'évaluation** : questions + morceau(x) correct(s).

Mesure	Définition
Rappel@k	part des questions dont un morceau correct est dans les k premiers
MRR	moyenne de 1/rang du premier morceau correct
Précision@k	part des k morceaux qui sont pertinents

Le concept

Mesurer la récupération (suite)

Si la récupération échoue, **aucun modèle ne peut répondre correctement** (sauf en devinant). Mesurer la récupération **séparément** de la génération.

Générer à partir des preuves

Consignes typiques : « Réponds **uniquement** à partir des extraits ci-dessous. Cite le numéro de l'extrait pour chaque affirmation. Si les extraits ne contiennent pas la réponse, réponds : *Je ne trouve pas cette information.* »

Même ainsi, le modèle peut mal lire, mélanger ou extrapoler : **vérifier** que chaque citation existe et soutient l'affirmation (mission 8).

Quand refuser

Le concept

Quand refuser (suite)

Un **seuil de confiance** : si le meilleur score de récupération est faible, ou si aucun extrait ne recoupe les mots essentiels de la question, **ne pas répondre** (ou poser une question de clarification). Refuser à tort coûte un peu ; répondre à tort coûte la confiance.

Pièges

- **Fuite entre utilisateurs** : un index partagé sans contrôle d'accès expose des documents confidentiels (mission 9).
- **Documents contradictoires ou périmés** : gérer versions et dates.
- **Instructions cachées dans des documents** (injection indirecte, mission 9).
- **Questions multi-étapes** : une seule récupération ne suffit pas (agents, mission 6).
- **Évaluation seulement « à l'œil »**.

Le concept

Pièges (suite)

🧠 ***Astuce*** *: regarde toujours les extraits récupérés avant de juger la réponse. Neuf fois sur dix, une mauvaise réponse vient d'une mauvaise récupération.*

Exemple pas à pas

Problème : construire un petit système de recherche (découpage, TF-IDF, BM25, vecteurs de mots), l'évaluer (rappel@k, MRR) en fonction de la taille des morceaux, fusionner les classements, produire une réponse **extractive avec citations**, et **refuser** quand la preuve est insuffisante.

```
import math, re
from collections import Counter

DOCUMENTS = {
 "remboursement": "Politique de remboursement. Un membre peut demander le rembours
   ement d'un abonnement dans un délai de 14 jours après l'achat, à condition de n
   e pas avoir terminé plus de 20 % du parcours. "
                  "Le remboursement est effectué sur le moyen de paiement d'origin
   e sous 5 jours ouvrés. Les achats de certificats ne sont pas remboursables.",
 "certificats": "Certificats. Un certificat est délivré lorsque tous les modules d
   'un parcours sont validés avec une note d'au moins 70 %. Le certificat est télé
   chargeable au format PDF et vérifiable en ligne grâce à un code unique. "
                "La réémission d'un certificat perdu coûte 5 euros.",
 "comptes": "Comptes. Pour changer de mot de passe, ouvrez Compte puis Sécurité. L
```

Exemple pas à pas (suite)

```
    'authentification à deux facteurs est recommandée. Un compte inactif pendant 24
     mois est supprimé après notification par e-mail.",
  "sessions": "Sessions en direct. Les sessions en direct ont lieu le mardi et le j
    eudi à 18 heures. L'enregistrement est disponible 24 heures après la fin de la
    session. Les questions peuvent être posées dans le forum du parcours.",
}
def mots(texte): return [m for m in re.findall(r"[a-zàâçéèêëîïôûùüÿœ0-9]+", texte.
    lower()) if len(m) > 2 and m not in {"les", "des", "une", "est", "pour", "que",
     "qui", "dans", "sur", "avec", "pas", "par", "aux", "son", "ses", "ont", "est"}
    ]

# ---------------- 1. découpage ----------------
def decouper(doc_id, texte, taille, chevauchement=0.15):
    phrases = [p.strip() for p in re.split(r"(?<=[.!?])\s+", texte) if p.strip()]
    morceaux, i = [], 0
    while i < len(phrases):
        courant, n = [], 0
        j = i
        while j < len(phrases) and (n + len(phrases[j].split()) <= taille or not c
```

Exemple pas à pas (suite)

```
   ourant): courant.append(phrases[j]); n += len(phrases[j].split()); j += 1
        morceaux.append((doc_id, " ".join(courant)))
        i = max(i + 1, j - max(0, round(len(courant) * chevauchement)))
    return morceaux
def indexer(taille): return [m for d, t in DOCUMENTS.items() for m in decouper(d,
   t, taille)]

# ---------------- 2. recherche lexicale : TF-IDF cosinus et BM25 ----------------
class Index:
    def __init__(self, morceaux):
        self.morceaux, self.tfs = morceaux, [Counter(mots(t)) for _, t in morceaux
   ]
        self.n = len(morceaux); self.longueurs = [sum(tf.values()) for tf in self.
   tfs]; self.moy = sum(self.longueurs) / self.n
        self.df = Counter(m for tf in self.tfs for m in tf)
    def idf(self, m): return math.log(1 + (self.n - self.df[m] + 0.5) / (self.df[m
   ] + 0.5))
    def bm25(self, question, k1=1.5, b=0.75):
        q = mots(question); scores = []
```

Exemple pas à pas

Exemple pas à pas (suite)

```
        for tf, L in zip(self.tfs, self.longueurs):
            scores.append(sum(self.idf(t) * tf[t] * (k1 + 1) / (tf[t] + k1 * (1 -
    b + b * L / self.moy)) for t in q if t in tf))
        return scores
     def tfidf_cosinus(self, question):
        vq = {t: c * self.idf(t) for t, c in Counter(mots(question)).items()}; nq
    = math.sqrt(sum(v * v for v in vq.values())) or 1
        scores = []
        for tf in self.tfs:
            vd = {t: c * self.idf(t) for t, c in tf.items()}; nd = math.sqrt(sum(v
     * v for v in vd.values())) or 1
            scores.append(sum(vq[t] * vd.get(t, 0) for t in vq) / (nq * nd))
        return scores
     def classement(self, scores): return sorted(range(self.n), key=lambda i: -scor
    es[i])

# ---------------- 3. jeu d'évaluation et mesures ----------------
EVAL = [("combien de jours pour demander un remboursement ?", "remboursement"), ("
    puis-je me faire rembourser un certificat ?", "remboursement"),
```

Exemple pas à pas (suite)

```
        ("quelle note faut-il pour obtenir un certificat ?", "certificats"), ("com
   bien coûte la réémission d'un certificat perdu ?", "certificats"),
        ("comment changer mon mot de passe ?", "comptes"), ("quand mon compte inac
   tif est-il supprimé ?", "comptes"),
        ("à quelle heure ont lieu les sessions en direct ?", "sessions"), ("où voi
   r l'enregistrement de la session ?", "sessions")]
def evaluer(taille, methode, k=1):
    idx = Index(indexer(taille)); rappel = mrr = 0
    for question, doc in EVAL:
        scores = idx.bm25(question) if methode == "bm25" else idx.tfidf_cosinus(qu
   estion)
        ordre = idx.classement(scores); docs = [idx.morceaux[i][0] for i in ordre]
        rappel += doc in docs[:k]; mrr += 1 / (docs.index(doc) + 1) if doc in docs
    else 0
    return rappel / len(EVAL), mrr / len(EVAL)
print("taille des morceaux | morceaux | BM25 rappel@1 / MRR | TF-IDF rappel@1 / MR
   R")
for taille in (12, 25, 60, 120):
    nb = len(indexer(taille)); b, t = evaluer(taille, "bm25"), evaluer(taille, "tf
```

Exemple pas à pas (suite)

```
   idf")
    print(f"       {taille:3d} mots       |   {nb:3d}    |    {b[0]:.2f} / {b[1]:.
   2f}     |    {t[0]:.2f} / {t[1]:.2f}")
assert evaluer(60, "bm25")[0] >= 0.75 and evaluer(60, "bm25")[1] >= 0.85

# ---------------- 4. fusion de rangs réciproques ----------------
def rrf(classements, k=60):
    score = Counter()
    for c in classements:
        for rang, i in enumerate(c): score[i] += 1 / (k + rang + 1)
    return [i for i, _ in score.most_common()]
idx = Index(indexer(25))
hybride_ok = 0
for question, doc in EVAL:
    fusion = rrf([idx.classement(idx.bm25(question)), idx.classement(idx.tfidf_cos
   inus(question))])
    hybride_ok += idx.morceaux[fusion[0]][0] == doc
print(f"fusion BM25 + TF-IDF : rappel@1 = {hybride_ok / len(EVAL):.2f}")
assert hybride_ok / len(EVAL) >= evaluer(25, "tfidf")[0]
```

Exemple pas à pas

Exemple pas à pas (suite)

```
# ---------------- 5. réponse extractive avec citations, ou refus ----------------
def repondre(question, k=2, seuil=3.0):
    idx = Index(indexer(25)); scores = idx.bm25(question); ordre = idx.classement(
   scores)[:k]
    if scores[ordre[0]] < seuil: return "Je ne trouve pas cette information dans l
   es documents.", []
    q = set(mots(question)); candidates = []
    for rang, i in enumerate(ordre, 1):
        doc, texte = idx.morceaux[i]
        for phrase in re.split(r"(?<=[.!?])\s+", texte):
            candidates.append((len(q & set(mots(phrase))), f"{phrase} [{rang}:{doc
   }]"))
    meilleure = max(candidates, key=lambda x: x[0])
    return meilleure[1], [(rang, idx.morceaux[i][0]) for rang, i in enumerate(ordr
   e, 1)]
for question in ("combien de jours pour demander un remboursement ?", "quel est le
    prix de la lune ?", "peut-on payer en bitcoin ?"):
    reponse, sources = repondre(question)
```

Exemple pas à pas (suite)

```
    print(f"Q : {question}\n   → {reponse}")
assert "14 jours" in repondre("combien de jours pour demander un remboursement ?")
   [0] and repondre("quel est le prix de la lune ?")[0].startswith("Je ne trouve p
   as")
```

Lecture :

1. Le **découpage** influe sur la récupération : des morceaux trop petits (12 mots) perdent le contexte ; des morceaux très grands mélangent plusieurs sujets. Le mieux se **mesure** avec le rappel@1 et le MRR sur un jeu de questions.
2. **BM25** et **TF-IDF cosinus** se comportent différemment selon les questions ; la **fusion de rangs réciproques** les combine sans calibrage de scores.
3. La réponse est **extractive** : on cite la phrase du document avec sa source (`[1:remboursement]`), donc on peut **vérifier**.

Exemple pas à pas (suite)

4. Quand le meilleur score est sous le **seuil**, le système **refuse** de répondre plutôt que d'inventer (« prix de la lune », « bitcoin »).

À toi, avec aide

À toi, avec aide

Un système de recherche a retourné, pour 6 questions, les classements suivants des morceaux (le morceau correct est indiqué) :

Question	Rang du premier morceau correct
Q1	1
Q2	3
Q3	1
Q4	non trouvé dans les 10 premiers
Q5	2
Q6	1

1. Calcule le rappel@1, le rappel@3 et le MRR (en considérant 0 pour « non trouvé »).

À toi, avec aide (suite)

2. Si l'on affiche k = 3 morceaux au modèle, quelle part des questions a au moins une chance d'être bien répondue (en supposant que le modèle sait lire un morceau correct) ?
3. Un second système obtient les rangs (1, 1, 2, 5, 1, 2). Lequel est meilleur selon chaque mesure ? Peut-on conclure avec 6 questions ?

Indices progressifs

Indice 1

MRR = moyenne de 1/rang ; 0 si non trouvé.

Indice 2

Rappel@k = part des questions dont le rang est ≤ k.

Indice 3

Avec 6 questions, l'incertitude est grande.

À toi, avec aide · réponse

Réponse

```
def mesures(rangs, ks=(1, 3)):
    n = len(rangs)
    mrr = sum(1 / r for r in rangs if r) / n
    return {k: sum(1 for r in rangs if r and r <= k) / n for k in ks}, mrr
A = [1, 3, 1, None, 2, 1]
B = [1, 1, 2, 5, 1, 2]
for nom, r in (("système A", A), ("système B", B)):
    rappels, mrr = mesures(r); print(f"{nom} : rappel@1 = {rappels[1]:.2f}, rappel
   @3 = {rappels[3]:.2f}, MRR = {mrr:.2f}")
assert round(mesures(A)[1], 2) == 0.64 and mesures(A)[0][1] == 0.5 and round(mesur
   es(A)[0][3], 2) == 0.83
```

1. Système A : **rappel@1 = 50 %**, **rappel@3 = 83 %**, **MRR = 0,64**.
2. Avec 3 morceaux affichés, **83 %** des questions ont le bon passage sous les yeux du modèle (au mieux).

Réponse (suite)

3. Système B : rappel@1 = 50 %, rappel@3 = 83 %, MRR ≈ 0,69 (rangs 1, 1, 1/2, 1/5, 1, 1/2). B trouve la réponse pour **toutes** les questions dans les 5 premiers, alors que A en manque une ; mais sur 6 questions, la différence n'est **pas statistiquement établie** : il faut un jeu d'évaluation plus grand (≥ 50 questions) et un intervalle de confiance (mission 7).

À toi, en solo

Système RAG évalué de bout en bout. Sur un corpus de 10 documents (à écrire, 100 à 200 mots chacun, sur HashBox) et 30 questions (dont 5 sans réponse dans le corpus) :

1. implémente BM25, TF-IDF cosinus et une recherche « vectorielle » avec des vecteurs de mots-clés regroupés par thèmes (un petit dictionnaire de synonymes) ;
2. évalue chaque méthode et leur fusion par RRF : rappel@1, rappel@3, MRR, avec intervalle bootstrap ;
3. choisis une taille de morceau en faisant varier de 10 à 150 mots ;
4. implémente le **seuil de refus** et mesure, pour 20 seuils, les deux erreurs : questions sans réponse **auxquelles on répond** (faux positifs) et questions avec réponse **que l'on refuse** (faux négatifs) ; trace la courbe et choisis un seuil avec une règle écrite d'avance ;
5. vérifie par code que chaque réponse extractive **cite** un morceau qui contient bien la phrase citée ;

À toi, en solo (suite)

6. rédige un rapport : forces et faiblesses de chaque méthode avec un exemple d'échec pour chacune.

À toi, en solo · réponse

Pistes de correction

```
import math, random, re
from collections import Counter

DOCS = {
 "remb1": "Remboursement. Un abonnement peut être remboursé dans les 14 jours suiv
   ant l'achat si moins de 20 % du parcours est terminé. Le remboursement se fait
   sur le moyen de paiement d'origine sous 5 jours ouvrés.",
 "remb2": "Achats non remboursables. Les certificats achetés séparément et les ses
   sions de mentorat déjà consommées ne sont pas remboursables.",
 "cert1": "Certificats. Un certificat est délivré après validation de tous les mod
   ules avec 70 % de bonnes réponses au minimum. Il est téléchargeable en PDF.",
 "cert2": "Réémission. Un certificat perdu peut être réédité pour 5 euros depuis v
   otre espace personnel.",
 "cpt1": "Mot de passe. Pour changer de mot de passe, ouvrez Compte puis Sécurité
   et suivez les instructions.",
 "cpt2": "Suppression. Un compte inactif depuis 24 mois est supprimé après notific
   ation par e-mail.",
 "live1": "Sessions en direct. Elles ont lieu le mardi et le jeudi à 18 heures, he
   ure de Paris.",
```

À toi, en solo · réponse

Pistes de correction (suite)

```
 "live2": "Enregistrements. L'enregistrement d'une session est publié 24 heures ap
   rès sa fin dans l'espace du parcours.",
 "mentor": "Mentorat. Un mentor est disponible 2 heures par semaine pour répondre
   aux questions sur les projets.",
 "prix": "Tarifs. L'abonnement mensuel coûte 29 euros ; l'abonnement annuel coûte
   290 euros.",
}
QUESTIONS = [("combien de jours pour être remboursé ?", "remb1"), ("le rembourseme
   nt prend combien de temps ?", "remb1"), ("peut-on rembourser un certificat ache
   té ?", "remb2"),
             ("quel pourcentage de bonnes réponses pour le certificat ?", "cert1")
   , ("comment récupérer un certificat perdu ?", "cert2"), ("changer le mot de pas
   se", "cpt1"),
             ("quand un compte inactif est-il supprimé ?", "cpt2"), ("heure des se
   ssions en direct", "live1"), ("où trouver l'enregistrement ?", "live2"), ("comb
   ien d'heures de mentorat ?", "mentor"),
             ("quel est le prix de l'abonnement annuel ?", "prix"), ("prix mensuel
   ", "prix")] * 2 + [("quelle est la capitale de la France ?", None), ("peut-on p
   ayer en bitcoin ?", None), ("qui est le président de HashBox ?", None), ("quel
```

Pistes de correction (suite)

```
    temps fera-t-il demain ?", None), ("combien pèse la lune ?", None)] + [("commen
    t obtenir un remboursement ?", "remb1")] * 1
def mots(t): return [m for m in re.findall(r"[a-zàâçéèêëîïôûùüÿœ0-9]+", t.lower())
     if len(m) > 2 and m not in {"les", "des", "une", "est", "pour", "que", "qui",
    "dans", "sur", "avec", "par", "aux"}]
SYN = {"rembours": "rembours", "remboursement": "rembours", "rembourser": "rembour
    s", "remboursé": "rembours", "prix": "prix", "coûte": "prix", "tarifs": "prix",
     "euros": "prix", "mot": "mdp", "passe": "mdp"}
def racine(m): return SYN.get(m, m[:6])
class Idx:
    def __init__(self):
        self.ids = list(DOCS); self.tf = {i: Counter(mots(t)) for i, t in DOCS.ite
    ms()}; self.n = len(DOCS); self.L = {i: sum(c.values()) for i, c in self.tf.ite
    ms()}; self.moy = sum(self.L.values()) / self.n
        self.df = Counter(m for c in self.tf.values() for m in c); self.tfs = {i:
    Counter(racine(m) for m in mots(t)) for i, t in DOCS.items()}; self.dfs = Count
    er(m for c in self.tfs.values() for m in c)
    def bm25(self, q, k1=1.5, b=0.75):
        out = {}
```

Pistes de correction (suite)

```
        for i in self.ids:
            s = 0
            for t in mots(q):
                if t in self.tf[i]: idf = math.log(1 + (self.n - self.df[t] + 0.5)
    / (self.df[t] + 0.5)); f = self.tf[i][t]; s += idf * f * (k1 + 1) / (f + k1 *
   (1 - b + b * self.L[i] / self.moy))
            out[i] = s
        return out
    def semantique(self, q):
        out = {}
        for i in self.ids:
            s = 0
            for t in {racine(m) for m in mots(q)}:
                if t in self.tfs[i]: s += math.log(1 + self.n / self.dfs[t]) * (1
   + math.log(self.tfs[i][t]))
            out[i] = s
        return out
def classer(sc): return sorted(sc, key=lambda i: -sc[i])
def rrf(*cl, k=60):
```

À toi, en solo · réponse

Pistes de correction (suite)

```
    s = Counter()
    for c in cl:
        for r, i in enumerate(c): s[i] += 1 / (k + r + 1)
    return [i for i, _ in s.most_common()]
idx = Idx(); repondables = [(q, d) for q, d in QUESTIONS if d]
def metriques(methode):
    r1 = r3 = mrr = 0; rangs = []
    for q, d in repondables:
        c = methode(q); rg = c.index(d) + 1; rangs.append(rg); r1 += rg == 1; r3 +
   = rg <= 3; mrr += 1 / rg
    n = len(repondables); return r1 / n, r3 / n, mrr / n, rangs
METHODES = {"BM25": lambda q: classer(idx.bm25(q)), "sémantique (racines + synonym
   es)": lambda q: classer(idx.semantique(q)), "RRF": lambda q: rrf(classer(idx.bm
   25(q)), classer(idx.semantique(q)))}
print("méthode                          | rappel@1 | rappel@3 | MRR  | IC95 bootst
   rap du MRR")
for nom, f in METHODES.items():
    r1, r3, mrr, rangs = metriques(f); rng = random.Random(1)
    boot = sorted(sum(1 / rng.choice(rangs) for _ in rangs) / len(rangs) for _ in
```

Pistes de correction (suite)

```
   range(1000))
    print(f"{nom:32s} |  {r1:.2f}    |  {r3:.2f}    | {mrr:.2f} | [{boot[25]:.2f}
   ; {boot[975]:.2f}]")
# seuil de refus
def erreurs(seuil):
    fp = fn = 0
    for q, d in QUESTIONS:
        s = max(idx.bm25(q).values()); repond = s >= seuil
        if d is None and repond: fp += 1
        if d is not None and not repond: fn += 1
    return fp / sum(1 for _, d in QUESTIONS if d is None), fn / len(repondables)
print("seuil | répond à tort (hors corpus) | refuse à tort (dans le corpus)")
courbe = {}
for seuil in (0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0):
    courbe[seuil] = erreurs(seuil); print(f" {seuil:4.1f}  |           {courbe[seu
   il][0]:5.0%}            |          {courbe[seuil][1]:5.0%}")
choisi = next(s for s in sorted(courbe) if courbe[s][0] <= 0.2 and courbe[s][1] <=
    0.15) if any(courbe[s][0] <= 0.2 and courbe[s][1] <= 0.15 for s in courbe) els
   e None
```

À toi, en solo · réponse

Pistes de correction (suite)

```
print("seuil retenu (≤ 20 % de fausses réponses ET ≤ 15 % de refus injustifiés) :"
   , choisi)
assert metriques(METHODES["RRF"])[2] >= 0.85
```

La **fusion** est au moins aussi bonne que chaque méthode prise séparément ; la **courbe de seuil** montre le compromis inévitable entre répondre à tort et refuser à tort — le seuil se choisit selon le **coût des deux erreurs** pour l'usage visé (support client : refuser coûte peu ; conseil juridique : répondre à tort coûte beaucoup).

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Juger la réponse sans regarder les extraits	On corrige le prompt alors que la recherche a échoué	Mesurer la récupération séparément (rappel@k, MRR)
Morceaux trop petits ou trop grands	Perte de contexte ou bruit	Tester plusieurs tailles et mesurer
Toujours répondre	Hallucination quand l'information manque	Seuil de confiance et réponse « je ne sais pas »
Index partagé sans contrôle d'accès	Fuite de documents confidentiels	Filtrer par droits avant la récupération

Mini-boss

Mini-boss

Assistant documentaire de HashBox avec évaluation complète.

1. corpus de 15 documents (politiques, tutoriels, FAQ) avec métadonnées (titre, date, niveau d'accès) ;
2. découpage par sections avec chevauchement, métadonnées conservées ;
3. recherche hybride (BM25 + vecteurs de racines/synonymes + RRF) avec filtre par droits d'accès appliqué **avant** la récupération ;
4. génération **extractive** avec citations ; refus sous seuil ; clarification quand deux documents se contredisent (dates différentes → choisir le plus récent et le dire) ;
5. jeu d'évaluation de 40 questions (dont 8 sans réponse, 4 sur documents contradictoires, 4 sur documents à accès restreint) ;
6. métriques : rappel@k, MRR, taux de refus justifié, taux de réponse à tort, fuite de documents restreints (doit être 0) ;

Mini-boss

Mini-boss (suite)

7. vérification automatique des citations (la phrase citée existe dans le morceau cité) ;
8. rapport : 3 échecs analysés (récupération ou lecture) et améliorations testées avec effet mesuré.

Mini-boss · réponse

Pistes de correction

Un bon travail : pipeline complet testé sur 40 questions; fuite de documents restreints mesurée à 0 grâce au filtre avant récupération; arbitrage entre documents contradictoires par date; vérificateur de citations qui rejette toute phrase absente du morceau; rapport avec échecs analysés (par exemple une question utilisant un synonyme absent du vocabulaire BM25, corrigée par un dictionnaire de synonymes ou la fusion vectorielle).

Quiz éclair

Quiz éclair

1. Le RAG consiste à :

A. rechercher des passages pertinents dans vos documents et les fournir au modèle pour qu'il réponde à partir d'eux
B. ré-entraîner le modèle à chaque question
C. désactiver le modèle

2. Le MRR est :

A. le nombre de documents
B. la moyenne de 1/rang du premier résultat correct
C. la longueur moyenne des réponses

3. Quand aucun extrait récupéré ne soutient la question, un bon système doit :

Quiz éclair

Quiz éclair (suite)

A. inventer la réponse la plus plausible
B. répondre quand même avec assurance
C. refuser de répondre ou demander une précision

4. 🔁 *Révision mission 03* : La fenêtre de contexte d'un modèle contient :

A. l'entrée et la sortie, en jetons, avec une limite totale
B. uniquement la question
C. uniquement la réponse

5. 🔁 *Révision mission 03* : Pour conserver un identifiant exact lors d'une compression d'historique, on :

A. le fait résumer par le modèle

Quiz éclair

Quiz éclair (suite)

B. l'épingle textuellement hors du résumé

C. le supprime

Quiz éclair · réponse

Réponses

1. **A**.
2. **B**.
3. **C**.
4. **A**.
5. **B**.

Je réfléchis

Je réfléchis

1. Dans ton organisation, quels documents contiennent la réponse à 80 % des questions répétitives, et sont-ils à jour ?
2. Que ferait ton système si deux documents se contredisaient ?

À retenir

À retenir

1. RAG : découper, indexer, récupérer, assembler, générer avec citations.
2. TF-IDF, BM25 (lexical), vecteurs (sémantique), RRF pour fusionner.
3. Mesurer la récupération : rappel@k, MRR, sur un jeu d'évaluation.
4. Réponse extractive avec citations vérifiables ; seuil de refus.
5. Droits d'accès avant la récupération ; documents périmés ou contradictoires.

MISSION ACCOMPLIE

+100 XP

Badge débloqué : Bibliothécaire de l'IA

Prochaine mission : Tool use et MCP

