1. VISION TECHNIQUE DE LA SOLUTION

1.1 Problématique Résolue

Notre solution RAG hybride répond au besoin critique des compagnies d'assurance de fournir un assistant intelligent capable de :

- **Comprendre** les questions complexes des clients en langage naturel
- **Accéder** aux données personnelles de manière sécurisée si l'utilisateur demande une information personnalisée (contrats, devis..) dans un site web vitrine
- **Générer** des devis auto-personnalisés en temps réel

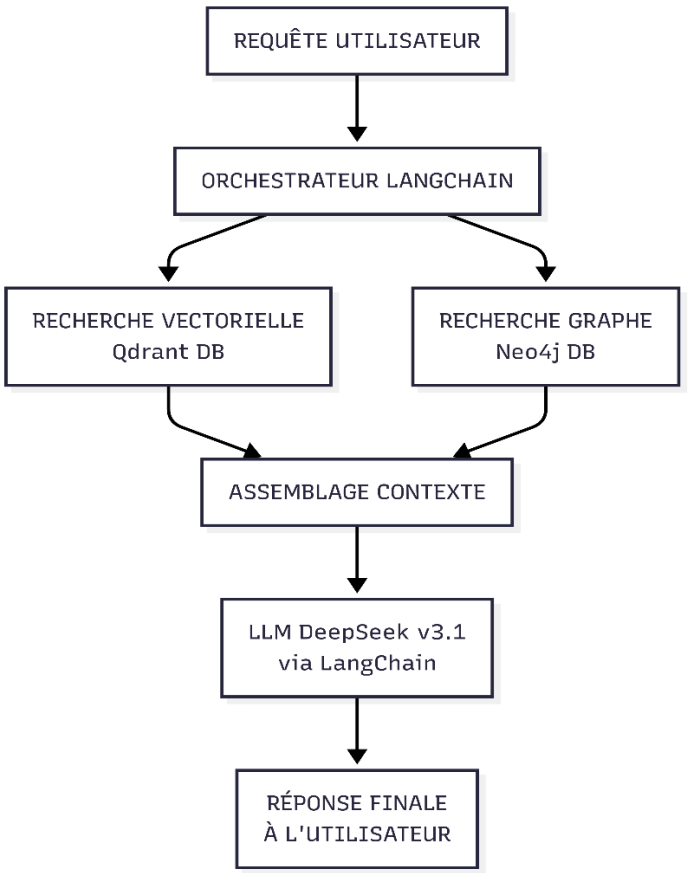
Notre système repose sur une **orchestration hybride et intelligente** qui combine quatre composantes clés :

- **Recherche Sémantique (Qdrant)**
  Exploite des embeddings vectoriels des données pour comprendre le sens des questions et retrouver les passages les plus pertinents dans les documents.
- **Recherche Relationnelle (Neo4j)**
  Explore les graphes de connaissances des données afin de révéler les relations entre entités, contrats, garanties et sinistres, offrant ainsi une vue explicative et connectée.
- **Génération Intelligente (DeepSeek v3.1)**
  Utilise un LLM open-source performant pour générer des réponses naturelles, précises et adaptées au contexte hybride (vectoriel + graphe).
- **Orchestration (LangChain)**
  Coordonne l'ensemble du pipeline : interrogation parallèle, fusion des résultats, et production de réponses enrichies et transparentes.

□ 2. ARCHITECTURE TECHNIQUE DÉTAILLÉE

2.1 Architecture Data Extraction :

2.2 Architecture RAG Hybride avec LangChain :



3. ÉTAPES TECHNIQUES DÉTAILLÉES DE LA SOLUTION

3.1 Phase 1 : Préparation et Structuration des Données

Le défi était que la majorité des données se présentaient sous forme de PDF contenant des images scannées (en français et en arabe), souvent de qualité médiocre. Un OCR classique n'était donc pas une solution optimale. J'ai donc opté pour une autre approche.

Étape 1 : Initialisation et Validation

> **Objectif : Vérifier que le fichier PDF existe et préparer la base de données.**
>
> - **Action 1.1** : Vérifier l'existence du fichier PDF.
> - **Action 1.2** : Initialiser la base de données PostgreSQL et créer les tables nécessaires (documents, pages_content).
> - **Résultat** : Le système est prêt à traiter le PDF.

Étape 2 : Conversion PDF → Images

> **Objectif : Transformer chaque page du PDF en image haute résolution.**
>
> - **Action 2.1** : Ouvrir le document PDF avec **PyMuPDF**.
> - **Action 2.2** : Parcourir chaque page et la convertir en image **PNG haute qualité (zoom x2)**.
> - **Action 2.3** : Sauvegarder les images avec un nom unique (page_001.png, page_002.png, …).
> - **Résultat** : Dossier contenant toutes les pages du PDF sous forme d'images prêtes pour OCR.

Étape 3 : Extraction du Texte (OCR)

> **Objectif : Extraire le texte présent dans les images via OCR.**
>
> - **Action 3.1** : Lire chaque image et l'encoder en **Base64** pour envoi à l'API.
> - **Action 3.2** : Envoyer l'image à l'API **Qwen-VL (via OpenRouter)** pour extraction de texte.
> - **Action 3.3** : Nettoyer et valider le texte renvoyé.
> - **Résultat** : Texte brut extrait de chaque page, stocké temporairement en mémoire.

Étape 4 : Stockage en Base de Données

> **Objectif : Sauvegarder les textes extraits avec leurs métadonnées.**
>
> - **Action 4.1** : Enregistrer le document dans la table documents (nom, chemin, nombre de pages).
> - **Action 4.2** : Enregistrer le texte de chaque page dans la table pages_content avec son numéro de page.
> - **Résultat** : Données textuelles disponibles et organisées dans PostgreSQL.

Étape 5 : Finalisation

> **Objectif : Vérifier que le processus s'est bien déroulé et libérer les ressources.**
>
> - **Action 5.1** : Logger le résultat (nombre de pages traitées, nombre total de caractères extraits).
> - **Action 5.2** : Fermer les fichiers PDF et la connexion à la base de données.
> - **Résultat** : Pipeline terminé avec succès, données stockées et traçabilité garantie.

3.1.1 Extraction et Vectorisation depuis PostgreSQL

Le but est de transformer le texte en vecteurs numériques qui capturent la sémantique. Permettre la recherche rapide et flexible via similarité vectorielle.

Étapes :

1. **Extraction** : récupérer les données de la BD.
2. **Segmentation** : découper le texte en chunks.
3. **Vectorisation** : générer des embeddings qui capturent le sens du texte.
4. **Indexation** : stocker les vecteurs dans **Qdrant** pour la recherche sémantique.

Technologies clés : PostgreSQL · psycopg2 · LangChain TextSplitter · SentenceTransformers · Qdrant

3.1.2 Construction du Graphe de Connaissances (Neo4j)

Le but est de créer un graphe structuré qui relie les informations contenues dans chaque document et chaque branche de documents. Ce graphe servira à répondre à des questions relationnelles ou à faire des comparaisons entre documents ou branches.

Le graphe est complémentaire à l'index vectoriel : le vectoriel capture le sens sémantique, le graphe capture les relations explicites entre entités et concepts.

Étapes :

1. **Identification des entités** : clients, contrats, garanties, sinistres, branches…
2. **Détection des relations** : *"UN CLIENT possède PLUSIEURS CONTRATS"*, *"UN CONTRAT inclut DES GARANTIES"*.
3. **Création du graphe** : insertion des nœuds et relations dans **Neo4j**.
4. **Optimisation** : création d'index sur les propriétés fréquemment interrogées.

Technologies clés : Neo4j · Cypher · spaCy (NER) · NetworkX

3.2 Phase 2 : Moteur de Recherche Hybride

3.2.1 Recherche Sémantique (Vector Search - Qdrant)

Objectif : comprendre le sens des requêtes et retrouver les documents les plus pertinents.

Étapes :

1. Générer un embedding de la requête utilisateur.
2. Comparer ce vecteur à ceux stockés dans Qdrant (similarité cosinus).
3. Classer les documents par pertinence.
4. Filtrer les résultats selon le contexte (branche, date, etc.).

3.2.2 Recherche Relationnelle (Graph Search - Neo4j)

Objectif : explorer les connexions entre entités pour des réponses contextuelles.

Étapes :

1. Identifier les entités présentes dans la requête.
2. Interroger Neo4j avec des requêtes Cypher ciblées.
3. Explorer les chemins relationnels (client → contrat → sinistre).
4. Évaluer la pertinence des chemins trouvés.

3.4 Phase 4 : Génération de Réponses Intelligentes

3.4.1 Assemblage du Contexte

Objectif : combiner toutes les infos utiles (vector + graph).

Étapes :

1. Récupérer les résultats vectoriels (infos générales).
2. Ajouter les résultats graphe si la requête est personnalisée.
3. Fusionner et supprimer les doublons.
4. Classer par pertinence et importance.

3.4.2 Génération avec LLM (DeepSeek v3.1)

Objectif : produire une réponse claire, naturelle et précise.

Étapes :

1. Construire un prompt (requête + contexte + intention).
2. Générer la réponse avec DeepSeek.
3. Nettoyer et valider la sortie (qualité, cohérence).

Pipeline d'Accès aux Données Personnelles

Quand un utilisateur demande ses informations personnelles (ex: "Je veux voir mes contrats", "Montrez-moi mes devis"), le système déclenche un processus d'authentification sécurisé en plusieurs étapes :

Étape 1 : Détection de l'intention "PERSONAL_DATA_ACCESS" via LangChain et extraction des entités.

Étape 2 : Vérification des informations d'identité et récupération de l'email dans PostgreSQL.

Étape 3 : Génération et envoi d'un code de vérification sécurisé (6 chiffres) par email.

Étape 4 : Stockage du code temporaire dans Redis (expiration 10 min).

Étape 5 : Validation du code et récupération des données personnelles via Neo4j.

Étape 6 : Formatage et présentation structurée à l'utilisateur, avec attribution des sources.

3.6 Phase 6 : Génération de Devis Intelligente

Étapes :

1. Extraction des paramètres (véhicule, conducteur, garanties…).
2. Appel de l'API de calcul de devis.
3. Enrichissement et présentation du devis (tarif, options, détails).