Système de Vérification Automatique de Conformité des Lignes PCR via LLM

1 Contexte du Projet

Dans le cadre des processus qualité ou métiers liés à l'échange de données des systèmes informatisés (bases de données transactionnelles, etc.), les données échangées entre entités (clients, partenaires, internes) suivent des formats stricts appelés **Plans de Contrôle de Référence (PCR)**.

Un PCR est un fichier texte contenant des lignes normalisées, chacune représentant :

- une transaction
- un objet métier
- une entité métier (ex : client, produit, commande)

Ces lignes doivent être conformes aux contraintes définies dans le fichier Word de spécifications. Historiquement, cette vérification se fait manuellement, par lecture croisée du fichier Word et du fichier texte. .

Exemple :

Spécification (Word)	**Fichier PCR (TXT)**
Tableaux décrivant : • Champs attendus • Leur ordre • Leur format (longueur, type) • Contraintes supplémentaires • Préfixe obligatoire	Lignes comme : `CLT12345678X JOHN DOE 0123456789`

2 Solution Proposée

Système de vérification automatique via LLM avec interface utilisateur en Streamlit combinant :

- Extraction des données
- Détection intelligente de préfixes
- Vérification structurée de comformité

3 Étapes Détaillées

3.1 Extraction des blocs de spécifications

Fonction : `get_table_paragraph_context_with_data(docx_path)`

Lire un fichier .docx et extraire :

- Paragraphe de contexte introduisant chaque tableau
- Tableau contenant les champs à extraire

Méthode : Utilisation de `python-docx` pour itérer sur les éléments avec `iter_block_items`. Chaque `Paragraph` suivi d'un `Table` est regroupé comme un bloc structuré avec un `block_index`.

Remarque : J'ai choisi cette méthode de structuration car les LLMs (LLaMA, Mistral ou Gemini) ne comprennent pas bien la structure native d'un tableau surtout s'ils sont transmis sous forme de texte tabulé ou de simples chaînes de caractères. . En effet, les modèles ont souvent du mal à extraire les relations logiques entre colonnes et lignes lorsque les données sont fournies sous une forme tabulaire brute. Pour pallier cela, j'ai transformé chaque ligne du tableau en dictionnaire Python, ce qui revient à représenter les données sous forme de JSON(Les LLM ont été massivement entraînés sur du JSON,).

Ce format est à la fois :

- Standardisé : le même pour tous les blocs, quelles que soient les spécifications.
- Lisible par le LLM : chaque champ devient une paire clé-valeur, facile à interpréter.
- Exploitant un contexte clair : le paragraphe d'introduction est attaché directement au tableau qu'il décrit, grâce à un block_index unique.

Ainsi, chaque bloc est constitué d'un paragraphe explicatif + un tableau structuré sous forme de dictionnaires, identifiés par leur block_index. Ce choix permet de maximiser la précision des détections, la compréhension des contraintes, et de simplifier le traitement automatique dans le prompt du LLM.

3.2 Détection intelligente du préfixe (LLM)

Fonctions :

- `build_prompt_json_ready(block_json)`
- `detect_prefix_llama_direct(block)`
- `run_prefix_detection_on_doc(docx_path)`

Identifier pour chaque bloc (= paragraphe+ tabeau du document word) le préfixe attendu , utilisé pour reconnaître à quel bloc appartient une ligne PCR donnée.

Challenge : Le préfixe peut être :

- Explicite (écrit littéralement dans le paragraphe ou le format attendu)
- Implicite (ex : "023 + 15 chiffres" → préfixe = "023")
- Caché dans une contrainte

Solution : Envoi de chaque bloc au modèle `meta-llama/Llama-4-Maverick-17B-128E-Instruct-FP8`

```
{
  "block_index": 0,
  "prefixe_detecte": "TRX"
}
```

Limite de TogetherAI ici : On attend 3 secondes entre chaque appel (rate limit = 6 000 tokens/minute), sinon les appels échouent. Le modèle peut aussi retourner un JSON invalide s'il dépasse les limites de contexte ou si la consigne n'est pas suivie strictement.

3.3 Lecture des lignes PCR

Fonction : `extract_pcr_lines(txt_path)`

```
TRX20240015DJOHNDOE5500EUR20240701
```

3.4 Appariement ligne PCR spécification

Fonction : `match_pcr_lines_to_blocks_by_prefix(blocks_with_prefixes, txt_path)`

Méthode : On teste `line.startswith(prefix)` pour chaque ligne et bloc pour associer chaque ligne du fichier PCR au bloc de spécification correspondant via le préfixe détecté précédemment. .

3.5 🔍 Vérification de conformité (LLM)

Fonctions :

- `build_conformity_prompt(line, matched_block)`
- `verify_conformity_with_llm(blocks_with_prefixes, txt_path)`

Objectif : Analyser si une ligne PCR respecte toutes les règles définies par son bloc associé (structure, contraintes, ordre, longueurs, etc.).

Détail du traitement :

- On génère un prompt avec `build_conformity_prompt`, contenant :
  - la ligne PCR
  - les champs de la spécification
- Le prompt est envoyé à un LLM Mistral :
  - `mistralai/Mixtral-8x7B-Instruct-v0.1` via TogetherAI

Réponse attendue : Un objet JSON structuré contenant :

- conformité globale de la ligne
- conformité de chaque champ :
  - nom
  - valeur
  - erreurs
  - longueur
- conformité de l'ordre des champs
- proposition d'une ligne corrigée si possible


```
{
  "line": "TRX20240015DJOHNDOE5500EUR20240701",
  "conforme": "non",
  "champs": [
    {
      "nom": "ID Transaction",
      "valeur": "TRX20240015",
      "conforme": "oui",
      "erreur": null,
      "longueur_attendue": 11
    },
    ...
  ],
  "ordre_champs": {
    "conforme": "oui",
```

```
    "ordre_attendu": ["ID", "Nom", "Montant"],
    "ordre_lu": ["ID", "Nom", "Montant"],
    "suggestion_ordre_corrige": null
  },
  "ligne_corrigee": "TRX20240015DOEJOHN5500EUR20240701"
}
```


3.6 Affichage des résultats dans Streamlit

Interface permettant de :

- Voir les préfixes détectés
- Consulter chaque ligne PCR et son statut
- Lire les erreurs champ par champ
- Télécharger un rapport final

Le diagramme illustre le workflow complet :

4 Limites et Améliorations

4.1 Limites rencontrées

- Taux d'échec élevé avec Together AI pour appels rapides
- Détection de préfixe et conformité peu fiables si les tokens dépasse le temps/nombre autorisé

4.2 Améliorations possibles

- Migration vers Google Gemini Pro (Token limit plus haute et 300$ offerts pour les nouveaux comptes)
- Envisager une facturation cloud (API Google Cloud, OpenAI, etc.)
- Fallback en demandant à l'utilisateur d'entrer manuellement le préfixe si le LLM échoue

Diagramme de Flux

Le diagramme illustre le workflow complet :

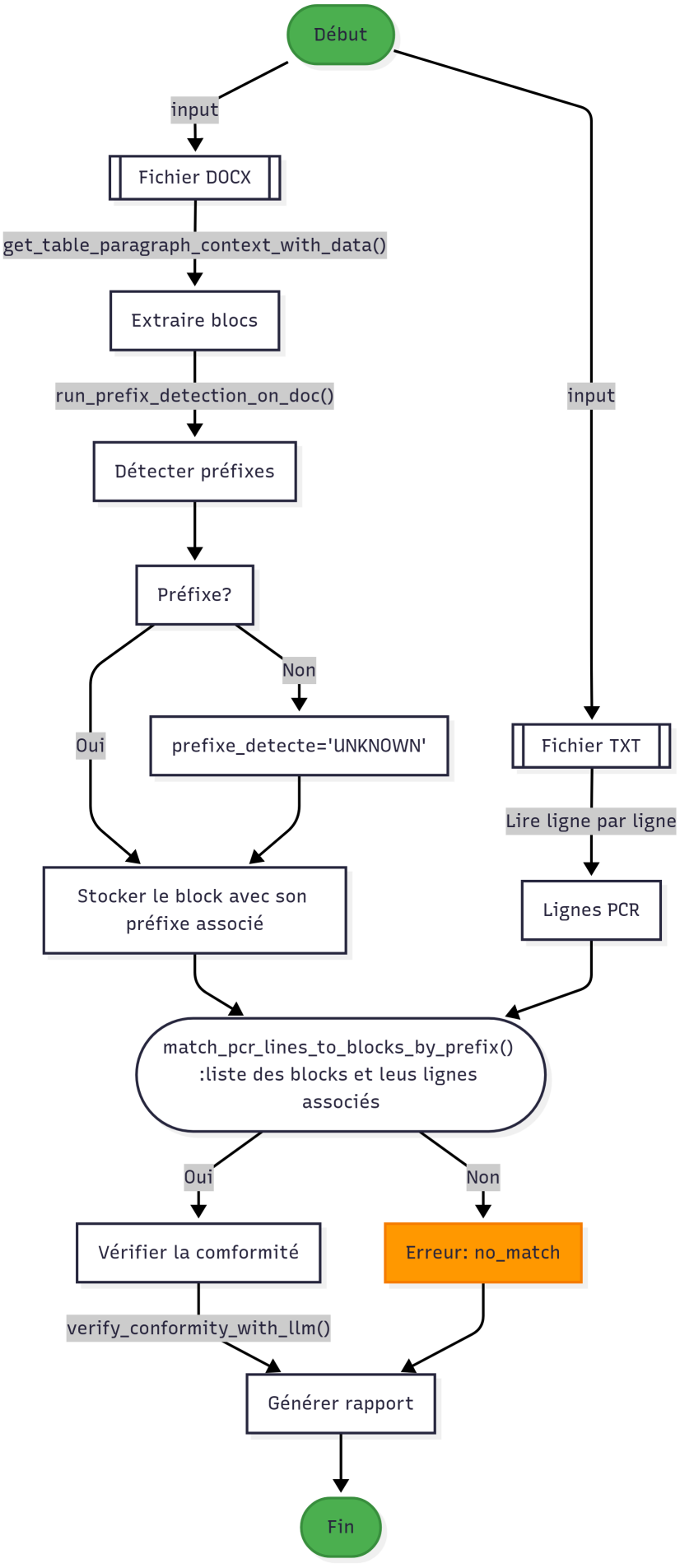

FIGURE 1 – Processus complet de vérification de conformité des lignes PCR