CY Tech

ING3 / IA Groupe B

Année 2025–2026

Natural Language Processing

Rapport de projet

Sujet : Systèmes Multi-Agents pour le NLP
Encadrant : Ismail Bouajaja

Auteurs :
Younes BENABDELLAH
Simon CHANTHRABOUTH-LIEBBE
David DEL CAMPO
Enzo EDMOND

Date : 31 janvier 2026

Résumé

Ce rapport présente une étude comparative entre un système mono-agent et un système multi-agents pour l'analyse automatisée de CVs dans le cadre d'un processus de recrutement. Le cas d'usage choisi est le tri de candidatures pour un poste de **Data Scientist Junior spécialisé en NLP**.

Le système multi-agents se compose de trois agents spécialisés travaillant en séquence : un **Agent Extracteur** qui structure les informations du CV en JSON, un **Agent Évaluateur** qui analyse l'adéquation avec le poste, et un **Agent Rédacteur** qui génère un email personnalisé. Le système mono-agent réalise l'ensemble de ces tâches en un seul appel au LLM.

Les expériences sont menées sur un corpus de **30 CVs** issus d'un dataset Kaggle, annotés manuellement avec un ground truth. Les résultats montrent que le multi-agent obtient une meilleure accuracy (82.61% vs 79.31%) et precision (72.73% vs 66.67%), mais présente un taux d'erreurs JSON plus élevé (23% vs 3%) et une latence 3 fois supérieure. Le mono-agent atteint un recall parfait (100%) mais génère davantage de faux positifs.

Ces résultats mettent en évidence un compromis entre précision de classification et robustesse du système, ouvrant des perspectives d'amélioration par des mécanismes de retry et de validation de format.

Mots-clés

Systèmes multi-agents, NLP, LLM, LangChain, Ollama, Llama 3, Classification de CVs, Recrutement automatisé, Pipeline séquentielle, Extraction d'information, Évaluation de candidatures, Génération de texte, Benchmark, Accuracy, F1-score, Precision, Recall

Keywords

Multi-agent systems, NLP, LLM, LangChain, Ollama, Llama 3, CV Classification, Automated recruitment, Sequential pipeline, Information extraction, Candidate evaluation, Text generation, Benchmark, Accuracy, F1-score, Precision, Recall

Table des matières

1 Introduction

L'automatisation des processus de recrutement représente un enjeu majeur pour les entreprises confrontées à des volumes importants de candidatures. L'analyse manuelle de CVs est une tâche chronophage et sujette à des biais, ce qui motive le développement de solutions basées sur l'intelligence artificielle.

Les récentes avancées dans le domaine des **Large Language Models (LLMs)** ouvrent de nouvelles perspectives pour le traitement automatique du langage naturel. Ces modèles sont capables d'extraire des informations structurées à partir de textes non structurés, d'évaluer la pertinence de profils par rapport à des critères donnés, et de générer des réponses personnalisées.

Dans ce projet, nous nous intéressons à la comparaison de deux approches pour l'analyse automatisée de CVs :

— Une approche **mono-agent**, où un seul LLM réalise l'ensemble des tâches (extraction, évaluation, génération) en une seule passe.

— Une approche **multi-agents**, où plusieurs agents spécialisés collaborent de manière séquentielle, chacun étant responsable d'une sous-tâche spécifique.

L'architecture multi-agents présente plusieurs avantages théoriques : une meilleure spécialisation de chaque agent, une décomposition claire des responsabilités, et une potentielle amélioration de la qualité des résultats grâce à la structuration intermédiaire des données. Cependant, elle introduit également des défis : une latence accrue, une consommation de tokens plus importante, et des risques de propagation d'erreurs entre agents.

Notre objectif est d'évaluer empiriquement ces deux approches sur un cas d'usage concret : le tri de candidatures pour un poste de **Data Scientist Junior spécialisé en NLP**. Nous utilisons le framework **LangChain** avec le modèle **Llama 3 (8B)** exécuté localement via **Ollama**, permettant une reproductibilité complète sans dépendance à des APIs externes.

Le rapport est organisé comme suit : la section 2 présente les données utilisées, la section 3 décrit les architectures mono et multi-agents, la section 4 détaille le protocole expérimental, la section 5 présente les résultats du benchmark, et la section 6 discute des limites et perspectives.

2 Données

2.1 Dataset de CVs

Le jeu de données utilisé provient de Kaggle [1] et contient des CVs au format PDF d'étudiants en écoles d'ingénieurs et masters, principalement dans les domaines de l'informatique, de la data science et du management.

Chaque CV est un fichier PDF contenant des informations variées : coordonnées, formation, expériences professionnelles, compétences techniques, langues et centres d'intérêt. La diversité des profils permet d'évaluer la capacité du système à discriminer les candidats pertinents pour un poste spécifique.

2.2 Description du poste cible

Le poste utilisé comme référence pour l'évaluation est défini comme suit :

```
Poste : Data Scientist Junior (NLP)
Recruteur : L'Equipe Super NLP

Competences requises :
- Python, pandas
- NLP (tokenization, embeddings, classification)
- Experience avec LLM / RAG est un plus

Experience :
- 0 a 2 ans (stage/alternance acceptes)

Diplome :
- Bac+5 (Master/ecole d'ingenieur) ou equivalent

Soft skills :
- Communication claire, rigueur
```

2.3 Ground Truth

Un fichier d'annotations `ground_truth.csv` a été créé manuellement pour 30 CVs, contenant pour chaque candidat :

- `cv_filename` : nom du fichier PDF
- `is_relevant` : 1 si le candidat devrait être convoqué en entretien, 0 sinon
- `relevance_score` : score de pertinence entre 0.0 et 1.0
- `justification` : courte justification de l'annotation

2.3.1 Critères d'annotation

Les critères suivants ont été appliqués pour déterminer la pertinence d'un candidat :

Candidat pertinent (is_relevant = 1) :

1. https://www.kaggle.com/datasets/anissamylaamri/cv-database-excel-pdf

— Maîtrise de Python (obligatoire)
— Compétences en NLP, Machine Learning ou Deep Learning
— Formation Bac+5 ou équivalent (école d'ingénieur, master)
— Expérience avec des outils comme TensorFlow, PyTorch, Scikit-learn, HuggingFace

Candidat non pertinent (is_relevant = 0) :
— Absence de Python dans les compétences
— Profil non technique (marketing, commerce, RH)
— Formation non adaptée ou trop junior (Bac+2/3)
— Orientation vers un autre domaine (énergie, génie civil, aéronautique)

2.3.2 Distribution des classes

Sur les 30 CVs annotés :
— **13 candidats pertinents** (43.3%) : profils Data Science, ML, NLP
— **17 candidats non pertinents** (56.7%) : profils hors cible

Cette distribution légèrement déséquilibrée reflète une situation réaliste où la majorité des candidatures ne correspondent pas exactement au poste recherché.

3 Architectures

3.1 Stack technologique

Les deux architectures (mono et multi-agents) partagent la même stack technologique :

— **LangChain** : framework Python pour la construction d'applications LLM

— **Ollama** : serveur local pour l'exécution de modèles LLM

— **Llama 3 (8B)** : modèle de langage open-source de Meta

— **PyPDF** : bibliothèque pour l'extraction de texte depuis les fichiers PDF

L'utilisation d'un modèle local permet une reproductibilité complète des expériences et évite les coûts associés aux APIs cloud.

3.2 Architecture Mono-Agent (Baseline)

L'architecture mono-agent constitue notre baseline. Elle utilise un seul appel au LLM pour réaliser l'ensemble des tâches :

1. Lecture du texte brut extrait du PDF
2. Analyse du CV par rapport à la description du poste
3. Génération d'une décision (entretien/refus) avec score et justification
4. Génération d'un email personnalisé

Le prompt fourni au modèle contient à la fois le CV complet et la description du poste, et demande une réponse structurée en JSON incluant tous les éléments nécessaires.

Avantages :

— Latence réduite (un seul appel LLM)

— Consommation de tokens optimisée

— Simplicité d'implémentation

Inconvénients :

— Pas de structuration intermédiaire des données

— Difficulté à isoler et corriger les erreurs

— Prompt complexe et long

3.3 Architecture Multi-Agents

L'architecture multi-agents décompose le processus en trois agents spécialisés travaillant en séquence :

3.3.1 Agent 1 : Extracteur

Rôle : Transformer le texte brut du CV en données structurées au format JSON.

Entrée : Texte extrait du PDF

Sortie : JSON contenant :

```
{
  "prenom": "...",
  "nom": "...",
```

```
  "email": "...",
  "telephone": "...",
  "diplomes": [...],
  "competences": [...],
  "experiences": [...]
}
```

3.3.2 Agent 2 : Évaluateur

Rôle : Analyser l'adéquation entre le CV structuré et la description du poste.

Entrées :

— JSON structuré du CV (sortie de l'Agent 1)

— Description du poste

Sortie : JSON contenant :

```
{
  "decision": "entretien" | "refus",
  "score_adequation": 0.0 - 1.0,
  "points_forts": [...],
  "points_faibles": [...],
  "justification": "..."
}
```

3.3.3 Agent 3 : Rédacteur

Rôle : Générer un email professionnel personnalisé basé sur la décision de l'évaluateur.

Entrées :

— Prénom et nom du candidat

— Décision et justification de l'Agent 2

Sortie : Email formaté (acceptation ou refus)

3.3.4 Workflow séquentiel

Le pipeline multi-agents suit un workflow strictement séquentiel :

$$\text{PDF} \xrightarrow{\text{PyPDF}} \text{Texte} \xrightarrow{\text{Agent 1}} \text{JSON CV} \xrightarrow{\text{Agent 2}} \text{Évaluation} \xrightarrow{\text{Agent 3}} \text{Email}$$

Chaque agent transmet sa sortie au suivant. Une erreur à n'importe quelle étape interrompt le pipeline.

Avantages :

— Spécialisation de chaque agent (prompts ciblés)

— Structuration intermédiaire facilitant le débogage

— Modularité (agents remplaçables indépendamment)

Inconvénients :

— Latence multipliée par le nombre d'agents

— Consommation de tokens accrue

— Risque de propagation d'erreurs (notamment JSON invalide)

4 Protocole expérimental

4.1 Configuration

Tous les tests sont réalisés avec la configuration suivante :

— **Modèle** : Llama 3 8B via Ollama
— **Température** : 0.2 (favorise la cohérence)
— **Nombre de CVs** : 30
— **Machine** : MacBook Air M4, 16GB RAM

4.2 Métriques d'évaluation

4.2.1 Métriques de classification

La tâche principale est une classification binaire : le candidat doit-il être convoqué en entretien (positif) ou non (négatif).

Accuracy :

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision :

$$\text{Precision} = \frac{TP}{TP + FP}$$

Proportion de candidats prédits "entretien" qui sont effectivement pertinents.

Recall :

$$\text{Recall} = \frac{TP}{TP + FN}$$

Proportion de candidats pertinents correctement identifiés.

F1-Score :

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

4.2.2 Mean Absolute Error (MAE)

Pour évaluer la qualité des scores d'adéquation prédits :

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

où y_i est le score du ground truth et $\hat{y}_i$ le score prédit.

4.2.3 Métriques de performance

— **Temps moyen par CV** : latence totale divisée par le nombre de CVs
— **Tokens totaux** : nombre de tokens consommés (entrée + sortie)
— **Taux d'erreurs** : proportion de CVs pour lesquels le pipeline échoue

4.3 Procédure de benchmark

1. Chargement de la liste des CVs depuis le dossier `data/kaggle_cvs/`
2. Chargement du ground truth depuis `data/ground_truth.csv`
3. Pour chaque CV :
   (a) Extraction du texte avec PyPDF
   (b) Exécution du pipeline mono-agent, mesure du temps et des tokens
   (c) Exécution du pipeline multi-agent, mesure du temps et des tokens
   (d) Comparaison des décisions avec le ground truth
4. Calcul des métriques agrégées
5. Export des résultats en CSV et JSON

5 Résultats expérimentaux

5.1 Résultats globaux

Le benchmark a été exécuté sur 30 CVs. Le tableau suivant présente les résultats comparatifs entre les deux approches :

TABLE 1 – Comparaison Mono-Agent vs Multi-Agent

Métrique	Mono-Agent	Multi-Agent
Accuracy	79.31%	**82.61%**
Precision	66.67%	**72.73%**
Recall	**100.00%**	88.89%
F1-Score	80.00%	80.00%
MAE Score	**0.150**	0.161
CVs évalués	29/30	23/30
Erreurs	1 (3.3%)	7 (23.3%)
Temps moyen/CV	**23.7s**	70.0s
Tokens totaux	**35,447**	60,171

5.2 Analyse des résultats

5.2.1 Classification

Le multi-agent obtient une meilleure **accuracy** (+3.3 points) et **precision** (+6 points), indiquant une meilleure capacité à identifier les vrais négatifs et à réduire les faux positifs.

Le mono-agent atteint un **recall parfait** (100%), ce qui signifie qu'il ne rate aucun candidat pertinent. Cependant, cette sensibilité élevée se fait au détriment de la précision : il génère davantage de faux positifs (accepte des candidats non pertinents).

Le **F1-score** est identique (80%) pour les deux approches, reflétant un équilibre différent entre precision et recall.

5.2.2 Robustesse

Le mono-agent présente un taux d'erreur de seulement 3.3% (1 erreur sur 30), contre 23.3% pour le multi-agent (7 erreurs). Les erreurs du multi-agent sont principalement dues à des réponses JSON invalides de l'Agent Extracteur.

5.2.3 Performance

Le mono-agent est **3 fois plus rapide** (23.7s vs 70.0s par CV) et consomme **40% moins de tokens** (35k vs 60k). Cette différence s'explique par le nombre d'appels LLM (1 vs 3) et la duplication d'informations entre agents.

5.3 Analyse des erreurs

5.3.1 Erreurs du Multi-Agent

Sur les 7 erreurs du multi-agent, l'analyse révèle les causes suivantes :

TABLE 2 – Types d'erreurs du Multi-Agent

Type d'erreur	Occurrences	Description
JSON invalide	6	Le LLM ajoute du texte avant/après le JSON ou utilise une syntaxe Python invalide
PDF illisible	1	PDF basé sur une image (scan), pas de texte extractible

Exemples d'erreurs JSON :
— Ajout de texte : `"Voici le CV au format JSON : {...}"`
— Syntaxe Python : `["Python" in competences]` au lieu de `["Python"]`
— JSON tronqué : réponse coupée avant la fin de l'objet

5.3.2 Faux positifs du Mono-Agent

Le mono-agent a incorrectement accepté 6 candidats non pertinents :
— **CV_Hiba_Errouhi** : Python + R mais profil généraliste
— **CV_Lakshay_Gupta** : Python Django mais backend, pas Data Science
— **CV_Logan_Marcellin-Dibon** : Python mais profil créatif (cinéma, musique)
— **CV_Melvil_Delahaye** : Python + R mais département énergie
— **CV_Romain_Trapp** : Python + DL mais département énergie
— **CV_Theo_Changarnier** : Profil IT/réseau, pas de Python explicite

Ces erreurs montrent que le mono-agent a tendance à être trop permissif lorsqu'il détecte des compétences techniques partiellement pertinentes.

5.3.3 Désaccords entre pipelines

Parmi les CVs évalués par les deux pipelines, on observe des désaccords notables :
— **CV_Charles_Sutty** : Mono=entretien, Multi=refus (GT=entretien)
→ Multi-agent trop strict malgré Python, ML, DL, Keras, TensorFlow
— **CV_Romain_Trapp** : Mono=entretien, Multi=refus (GT=refus)
→ Multi-agent correctement identifié comme profil énergie
— **CV_Rim_ElFatihi** : Mono=refus, Multi=entretien (GT=refus)
→ Multi-agent trop permissif (1ère année, trop junior)

6 Discussion

6.1 Synthèse des résultats

TABLE 3 – Récapitulatif des forces et faiblesses

Critère	Mono-Agent	Multi-Agent
Accuracy	−	+
Precision	−	+
Recall	+	−
Robustesse	+	−
Latence	+	−
Coût (tokens)	+	−
Interprétabilité	−	+

6.2 Interprétation

Pourquoi le multi-agent est plus précis ? La décomposition en agents spécialisés permet une meilleure structuration de l'analyse. L'Agent Évaluateur reçoit un CV déjà formaté en JSON, ce qui facilite la comparaison systématique avec les critères du poste. Le mono-agent doit simultanément extraire et évaluer, ce qui peut diluer son attention.

Pourquoi le mono-agent a un meilleur recall ? Le mono-agent adopte une stratégie plus "inclusive" : en cas de doute, il tend à accepter le candidat. Cette approche minimise les faux négatifs mais augmente les faux positifs.

Pourquoi le multi-agent a plus d'erreurs ? Le pipeline multi-agent est plus fragile car chaque agent doit produire une sortie dans un format strict (JSON). Le modèle Llama 3 8B, bien que performant, ne respecte pas toujours les contraintes de format, notamment lorsque le contexte est long ou ambigu.

6.3 Limites de l'étude

— **Taille du dataset** : 30 CVs est un échantillon limité pour tirer des conclusions statistiquement robustes.
— **Subjectivité du ground truth** : L'annotation manuelle introduit une part de subjectivité dans l'évaluation.
— **Modèle unique** : Les résultats sont spécifiques à Llama 3 8B ; d'autres modèles pourraient présenter des comportements différents.
— **Absence de retry** : Le benchmark ne tente pas de relancer les agents en cas d'erreur JSON, ce qui pénalise le multi-agent.
— **PDFs hétérogènes** : Certains CVs sont mal formatés ou basés sur des images, limitant l'extraction de texte.

6.4 Pistes d'amélioration

1. Mécanisme de retry avec validation JSON Implémenter un système de retry lorsque la réponse du LLM n'est pas un JSON valide. Le prompt de relance peut inclure le message d'erreur pour guider le modèle.

2. Extraction JSON robuste Utiliser des expressions régulières ou des parsers tolérants pour extraire le JSON même lorsqu'il est entouré de texte parasite.

3. Few-shot prompting Inclure des exemples de réponses correctes dans les prompts pour améliorer la conformité au format attendu.

4. Modèle plus performant Tester avec des modèles plus grands (Llama 3 70B) ou spécialisés dans le suivi d'instructions (Mistral, Claude).

5. Agents parallèles Pour certaines tâches indépendantes, exécuter les agents en parallèle pourrait réduire la latence globale.

7 Conclusion

Ce projet a permis de comparer deux architectures pour l'analyse automatisée de CVs à l'aide de LLMs : une approche mono-agent et une approche multi-agents à trois composants (Extracteur, Évaluateur, Rédacteur).

Principaux résultats :

- Le **multi-agent** obtient une meilleure accuracy (82.61% vs 79.31%) et precision (72.73% vs 66.67%), démontrant l'intérêt de la spécialisation des agents pour une classification plus discriminante.
- Le **mono-agent** présente un recall parfait (100%) et une robustesse supérieure (3% d'erreurs vs 23%), au prix d'une précision moindre.
- Le **F1-score** est identique (80%) pour les deux approches, reflétant des compromis precision/recall différents mais équivalents en termes de performance globale.
- Le mono-agent est **3 fois plus rapide** et consomme **40% moins de tokens**, ce qui en fait une solution plus économique pour des applications à grande échelle.

Recommandations :

- Pour une application où **aucun candidat pertinent ne doit être manqué** (minimiser les faux négatifs), le mono-agent est préférable grâce à son recall de 100%.
- Pour une application où la **qualité des présélections est critique** (minimiser les faux positifs), le multi-agent offre une meilleure precision.
- Dans les deux cas, l'ajout de mécanismes de **retry et validation** améliorerait significativement la robustesse.

Perspectives :

Ce travail ouvre plusieurs pistes d'amélioration : l'intégration d'un agent critique pour valider les décisions, l'utilisation de modèles plus performants, et l'extension à d'autres cas d'usage NLP (analyse de documents juridiques, support client automatisé). L'approche multi-agents pourrait également bénéficier de mécanismes de mémoire partagée et de planification dynamique pour des workflows plus complexes.

Références

[1] LangChain, "LangChain : Building applications with LLMs through composability," 2024. Available : `https://python.langchain.com/`.

[2] Ollama, "Ollama : Run Llama 3, Mistral, and other LLMs locally," 2024. Available : `https://ollama.com/`.

[3] Meta AI, "Llama 3 : Open foundation and instruction-tuned models," 2024. Available : `https://llama.meta.com/`.

[4] A. M. Laamri, "CV Database Excel PDF," Kaggle, 2024. Available : `https://www.kaggle.com/datasets/anissamylaamri/cv-database-excel-pdf`.

[5] T. Wu et al., "AutoGen : Enabling next-gen LLM applications via multi-agent conversation framework," *arXiv preprint*, arXiv :2308.08155, 2023.