Une architecture multi-agents pour la génération automatique de tickets en environnement industriel : focus sur l'agent de classification

Ying Zhang[1], Sébastien Bonnet[2], Matthieu Petit Guillaume[1],
Muriel Hug[1], Aurélien Krauth[1], Rémi Uhartegaray[2]

[1] Leviatan, 725 Bd Robert Barrier, 73100 Aix-les-Bains
[2] AntemetA, 19 Av. Georges Pompidou, Bâtiment Le Danica, 69003 Lyon

{y.zhang, matthieu, m.hug, aurelien}@leviatan.fr
{sebastien.bonnet, remi.uhartegaray}@antemeta.fr

Résumé

La génération automatique de tickets à partir d'e-mails constitue un enjeu majeur pour le support client en environnement industriel, où confidentialité et ressources limitées sont critiques. Nous proposons une architecture multi-agents composée d'un agent de classification pour identifier la catégorie de l'e-mail et d'agents d'extraction spécialisés pour collecter les informations requises. Dans cet article, nous présentons la conception, l'entraînement et l'évaluation de l'agent de classification, basé sur un grand modèle de langage open source quantifié en 4 bits et affiné via PEFT/LoRA. Les résultats montrent qu'avec environ 13 milliards de paramètres, il est possible d'atteindre plus de 88% d'exactitude tout en répondant aux contraintes industrielles. Cette approche modulaire ouvre la voie à une automatisation complète du processus, chaque agent étant dédié à une tâche spécifique du workflow.

Mots-clés

Classification d'e-mails, génération automatique de tickets, grands modèles de langage, fine-tuning, multi-agents, quantification 4 bits, industrie.

Abstract

Automatically generating tickets from e-mails is a key challenge for industrial customer support, especially under constraints of data confidentiality and limited resources. We propose a multi-agent architecture with a single classification agent that determines the ticket category and specialized extraction agents that gather the required fields. In this paper, we focus on the design, training, and evaluation of the classification agent, built upon an open-source large language model quantized to 4 bits and fine-tuned via PEFT/LoRA. Our results show that a mid-sized model (approximately 13 billion parameters) can achieve over 88% accuracy while complying with industrial constraints. This modular approach paves the way for fully automated workflows, with each agent dedicated to a specific stage of the ticket-generation pipeline.

Keywords

E-mail classification, automatic ticket generation, large language models, fine-tuning, multi-agent, 4-bit quantization, industry.

1 Introduction

Dans le secteur du support client, le nombre croissant d'e-mails reçus chaque jour pose d'importants défis aux équipes de maintenance et d'assistance. À partir de ces e-mails, des tickets doivent être créés. Ce processus consiste à trier, classer et affecter ces tickets aux équipes responsables en fonction de leur nature et de leur urgence, puis, selon les différentes catégories de tickets, à générer un fichier JSON dont la structure (attributs) varie, afin de l'intégrer directement dans notre outil de gestion.

Les processus manuels de lecture et de tri entraînent souvent des délais de traitement élevés, des erreurs de classification et une surcharge de travail. Face à ces contraintes, l'automatisation de la création de tickets à partir d'e-mails constitue donc un levier stratégique pour accroître la réactivité et la qualité du service.

Dans le cadre de notre projet, nous adoptons une approche multi-agents pour couvrir les différentes étapes de l'analyse et du traitement des tickets. Bien que ce système comporte plusieurs « agents », nous concentrerons ici notre attention sur l'agent de classification. L'objectif est de démontrer qu'un modèle de langage, correctement entraîné et adapté aux contraintes industrielles (sécurité des données et ressources matérielles limitées), peut catégoriser automatiquement les e-mails clients tout en maintenant un haut niveau de précision.

Après avoir présenté le contexte, la problématique et l'état de l'art, nous décrirons l'architecture multi-agents dans ses grandes lignes, puis nous focaliserons sur le fonctionnement, l'entraînement et l'évaluation de l'agent de classification. Enfin, nous discuterons des résultats obtenus et évoquerons les perspectives d'évolution.

2 Contexte et problématique

Dans un environnement où le support client doit traiter chaque jour un volume considérable d'e-mails, la gestion manuelle de ces messages devient rapidement un goulot d'étranglement. Les opérateurs doivent lire et comprendre chaque demande, puis la classer dans une catégorie de ticket appropriée (incident, demande de service, question d'ordre technique, etc.). Cette opération, très coûteuse en temps, reste par ailleurs sujette aux erreurs humaines, pouvant entraîner une mauvaise affectation des demandes et retarder la prise en charge de problèmes critiques.

L'émergence des grands modèles de langage (Large Language Models, LLM) ouvre la voie à une automatisation plus poussée de ce processus. Toutefois, dans un cadre industriel, l'adoption de ces modèles soulève des défis spécifiques :

- **Sécurité des données** : Les contenus échangés par e-mail peuvent être sensibles et ne doivent pas être exposés à des services publics.
- **Contraintes matérielles** : Déployer un LLM de très grande taille requiert des ressources (GPU, VRAM) considérables, souvent incompatibles avec l'infrastructure standard d'une PME (Petite et Moyenne Entreprise).
- **Fiabilité de la classification** : Le modèle doit offrir une précision suffisante pour que l'automatisation ne génère pas de nouvelles sources d'erreurs.

Ainsi, la problématique principale consiste à concilier les avantages offerts par les LLM (compréhension fine du langage, capacité à traiter rapidement de gros volumes de données) avec les contraintes d'un déploiement industriel (infrastructures matérielles limitées et données confidentielles), tout en préservant une qualité de classification à la hauteur des exigences d'un service client professionnel.

3 État de l'art

Ces dernières années, les avancées en traitement automatique du langage ont été largement portées par l'émergence des modèles de langage de grande taille (LLM). Ces modèles, tels que LLaMA [8], GPT [10], Qwen [12], et d'autres architectures émergentes, ont permis des améliorations notables dans des domaines comme la compréhension du langage, la génération de texte, et la traduction automatique.

Cependant, lorsqu'il s'agit d'implémenter ces modèles dans des environnements industriels, plusieurs défis apparaissent. Comme indiqué précédemment, la confidentialité des données et l'importante consommation de ressources matérielles constituent deux freins majeurs à l'adoption des LLM. Même un modèle puissant doit être adapté aux données métier (domain adaptation), ce qui implique un micro-ajustement (fine-tuning) potentiellement coûteux.

Pour faire face à ces contraintes, différentes stratégies d'optimisation sont proposées, comme la quantification (passage en int8, int4, etc.) [2, 6], ou les approches de Parameter-Efficient Fine-Tuning (PEFT) [7], telles que LoRA [5, 2], qui réduisent la taille des poids à ajuster et permettent de se rapprocher des performances d'un modèle complet. Parallèlement, l'approche multi-agents [13] en IA gagne en popularité, chaque agent se voyant confier une tâche spécialisée (classification, extraction d'information, etc.), ce qui facilite la modularité et l'évolutivité du système. Dans le cadre du présent travail, nous nous inscrivons dans cette continuité, en centrant l'analyse sur la partie « classification » pour répondre aux besoins de tri automatisé des e-mails.

4 Conception en multi-agents et focalisation sur l'agent de classification

Un large éventail de modèles open source a vu le jour, chacun offrant plusieurs versions de différentes tailles en nombre de paramètres. Cependant, il existe une différence de qualité notable entre les versions à faible et à fort nombre de paramètres : les modèles plus légers (quelques milliards de paramètres) sont plus rapides et moins gourmands en ressources, mais leurs performances en compréhension et en génération de texte sont souvent inférieures, en particulier pour les tâches complexes.

Dans notre environnement industriel, nous privilégions le déploiement de LLM d'environ 13 milliards de paramètres. Néanmoins, ce type de modèle reste moins performant qu'un modèle de 70 milliards [3], sans même parler de ceux qui comptent plusieurs centaines de milliards de paramètres. Cette différence exige des optimisations algorithmiques, une adaptation des données ainsi qu'un entraînement spécifique à chaque tâche pour compenser le manque de puissance brute.

Dans notre cas d'usage, le nombre de catégories de tickets peut dépasser les 300. Toutefois, nous avons choisi, dans un premier temps, de cibler uniquement certaines catégories principales. Chacune d'elles requiert l'extraction de champs d'informations spécifiques. Pour répondre à ces besoins variés, il serait peu judicieux de définir un unique « agent » chargé à la fois de la classification et de l'extraction de tous les champs, car le prompt deviendrait trop complexe et potentiellement ambigu. Pour un modèle de 13 milliards de paramètres, cela s'avérerait quasiment impossible.

Afin de mieux structurer notre système, nous avons opté pour une architecture multi-agents [13]. Concrètement :

- Un premier agent se consacre exclusivement à la classification des e-mails afin d'identifier la catégorie.
- Un agent d'extraction spécifique à chaque catégorie prend ensuite le relais pour extraire les champs requis en fonction de la catégorie identifiée.

Un orchestrateur central coordonne ces agents : dès qu'un e-mail est reçu, il est classé, puis redirigé vers l'agent d'extraction adéquat, si besoin. Ainsi, nous obtenons un total de $N+1$ agents, où N représente le nombre de catégories. Cette approche offre :

- **Modularité accrue** : chaque catégorie peut évoluer indépendamment (ajout de nouveaux champs à extraire, ajustement du prompt, etc.).

— **Lisibilité renforcée** : chaque agent d'extraction est axé sur un ensemble restreint d'informations à récupérer, ce qui améliore la qualité.
— **Maintenance facilitée** : en cas de nouvelle catégorie, il suffit d'ajouter un nouvel agent d'extraction et de réajuster l'agent de classification, sans perturber les autres agents.

Toutefois, au stade actuel du projet, seul l'agent dédié à la classification a atteint la maturité requise pour un déploiement et une évaluation rigoureuse. Les autres agents sont encore en phase d'expérimentation et feront l'objet de travaux futurs.

Dans ce contexte, le présent article se concentre exclusivement sur l'agent de classification, depuis la sélection et l'adaptation du modèle de langage (LLM) jusqu'à l'évaluation de ses performances. Ce choix nous permet de proposer une étude détaillée d'une solution déjà opérationnelle, tout en soulignant l'intérêt et la faisabilité de la démarche multi-agents dans un environnement industriel.

5 Jeu de données

L'ensemble de données utilisé dans cette étude provient de la base de gestion des tickets de support client. Son objectif principal est de permettre l'entraînement et l'évaluation d'un agent de classification des e-mails en fonction des catégories de tickets.

5.1 Structure et origine des données

Les données initiales comprenaient 175 813 entrées issues du système de gestion des tickets, chaque entrée étant caractérisée par les attributs suivants :

— **Numéro de ticket** : Identifiant unique pour chaque ticket.
— **Description** : Contenu des échanges de mails entre le client et le support technique.
— **Catégorie du ticket** : Classification du type de ticket (ex. : incident technique, facturation, demande de service, etc.).
— **Message** : Réponses du support envoyées aux clients, comprenant mises à jour et modifications du ticket.

L'objectif de cette classification repose sur l'analyse de la première communication client pour chaque ticket, afin d'identifier et de caractériser la nature initiale des demandes. Toutefois, l'étude des données brutes révèle des contraintes méthodologiques liées à l'absence d'un ensemble clairement défini de « premiers e-mails » envoyés par les clients, ce qui rend difficile l'extraction rigoureuse de la requête initiale. Par ailleurs, de nombreux tickets présentent déjà plusieurs échanges par e-mail avant même leur enregistrement, compromettant l'isolement de la première interaction et compliquant l'application de procédures de classification fiables.

5.2 Nettoyage des données

Pour assurer la qualité et la pertinence du jeu de données destiné à la classification, plusieurs étapes de préparation et de nettoyage ont été mises en œuvre :

1. Filtrage initial des colonnes
Les enregistrements ne respectant pas la structure à quatre colonnes prédéfinies ont été éliminés, réduisant l'ensemble à 154 414 lignes conformes.
2. Regroupement par Numéro de ticket
Les données ont été agrégées par Numéro de ticket afin de ne conserver que la première ligne associée à chaque identifiant, tandis que les lignes subséquentes pour un même ticket ont été supprimées. En effet, chaque enregistrement correspond à un état de mise à jour d'un ticket (par exemple : création, modification, fermeture, etc.), pouvant ainsi générer plusieurs enregistrements pour un même identifiant. Cette opération a permis de ramener l'ensemble de données à 29 376 tickets uniques.
3. Extraction des premiers e-mails
Étant donné que la classification repose sur l'analyse de la première communication reçue du client, un traitement supplémentaire a été appliqué à la colonne « Description » (où se trouvent souvent plusieurs échanges). Ce traitement visait à isoler précisément les informations correspondant au premier e-mail.
Une règle fondée sur la détection de motifs textuels récurrents (par exemple : « To… », « Sujet… », etc.) a été mise en place pour segmenter les différentes parties du fil de discussion et extraire la première communication du client.
Cette approche de segmentation reste rudimentaire et pourrait être optimisée dans de futurs travaux. Néanmoins, elle permet d'isoler de manière satisfaisante le texte relatif à la première prise de contact.
4. Analyse des catégories
Enfin, les fréquences des différentes catégories de tickets ont été étudiées afin de déceler d'éventuels déséquilibres de classe, susceptibles d'influencer la performance des algorithmes de classification.

5.3 Sélection des catégories

L'analyse des 29 376 tickets a mis en évidence 365 catégories distinctes. Une répartition inégale des occurrences a été constatée : la catégorie la plus fréquente regroupe 10 963 tickets, tandis que plus de 200 catégories ne comptent chacune que moins de cinq occurrences.

Afin de garantir la pertinence et la qualité des expérimentations, les catégories ont été soumises à un filtrage rigoureux selon deux critères :

1. Sélection de 24 catégories stratégiques
Ces catégories ont été jugées essentielles pour l'entreprise en raison de leur impact opérationnel et de leur récurrence.
2. Seuil minimal de représentativité de 2‰
Les catégories dont la proportion dans l'ensemble de données est inférieure à 2‰ ont été considérées comme sous-représentées et donc écartées de l'étude.

À l'issue de ce processus, neuf catégories ont été retenues pour la phase de recherche et développement (R&D) :

— Demande de service/Backup BCS/Autre
— Demande de service/Backup BCS/Demande de renseignement
— Demande de service/Backup BCS/Restauration qualifiée

— Demande de service/Backup BCS/Stratégie de sauvegarde/Création
— Demande de service/Backup BCS/Stratégie de sauvegarde/Modification
— Demande de service/Backup BCS/Stratégie de sauvegarde/Suppression
— Demande de service/Cyber Sécurité CS2/Bastion/Création-Modification d'entrées
— Incidents/Backup BCS/Sauvegarde
— Incidents/Supervision

Cette sélection vise à concentrer les efforts de modélisation sur les catégories les plus pertinentes, tant sur le plan stratégique qu'en termes de volume de données.

5.4 Constitution de l'ensemble de données final

L'ensemble final est composé de 4500 tickets, répartis entre un ensemble d'entraînement (80 %) et un ensemble de test (20 %), selon la distribution suivante :

Catégorie	Train Set	Test Set
Incidents/Supervision	2397	565
Demande de service/Backup #BCS/Restauration qualifiée	361	83
Incidents/Backup #BCS/Sauvegarde	346	95
Demande de service/Backup #BCS/Autre	200	52
Demande de service/Backup #BCS/Stratégie de sauvegarde/Création	78	21
Demande de service/Backup #BCS/Stratégie de sauvegarde/Suppression	69	25
Demande de service/Backup #BCS/Demande de renseignement	58	17
Demande de service/Backup #BCS/Stratégie de sauvegarde/Modification	50	14
Demande de service/Cyber Sécurité #CS2/Bastion/Création-Modification d'entrées	49	20

TABLE 1 – Répartition des tickets

L'ensemble de données ainsi préparé garantit :

— Une couverture des principales catégories rencontrées en support client.
— Un équilibre entre les différentes classes, tout en évitant les catégories sous-représentées.
— Une qualité de données optimisée, grâce à la sélection des premières interactions client et l'élimination des doublons et bruits textuels.

Ce jeu de données constitue une base fiable pour entraîner et évaluer un modèle de classification automatique des e-mails en environnement industriel. L'ensemble de nos entraînements a été réalisé sur ce jeu de données d'entraînement (train set), tandis que toutes les évaluations de benchmark ont été effectuées sur ce jeu de test (test set).

6 Configuration matérielle

Les expérimentations de cette tâche ont été réalisées sur une machine dotée des caractéristiques suivantes :

— GPU : 1 × NVIDIA A10 avec 24 Go de VRAM.
— CPU : 30 cœurs.
— RAM : 214,7 Go.
— Stockage : 1,5 To SSD.

Cette configuration permet de répondre aux exigences en termes de calcul intensif pour le benchmarking des modèles tout en offrant une capacité de stockage suffisante pour gérer les données expérimentales.

7 Benchmark basé sur les modèles initiaux

7.1 Ingénierie des prompts et des plateformes

Pour évaluer les performances initiales des modèles dans la tâche de classification, nous avons mis en place deux approches distinctes de benchmark, reposant sur des plateformes et des types de prompts différents.

La première approche utilise la plateforme Ollama [9], avec un prompt conçu pour générer une sortie au format JSON standardisé. Celui-ci est structuré de manière à inclure une explication de la tâche, une présentation des catégories, des règles spécifiques de classification, ainsi qu'un exemple annoté et une définition explicite du format JSON attendu. Cette méthode vise à assurer une sortie normalisée et directement exploitable. Un exemple détaillé du prompt est disponible en Annexe A.1.

La seconde approche repose sur la plateforme Unsloth [1], où le prompt est formulé en langage naturel, sans contrainte de structure JSON dans les résultats. Bien que sa conception s'appuie sur des principes similaires à ceux du prompt JSON, il privilégie une approche plus souple pour évaluer la capacité du modèle à comprendre et exécuter la classification de manière plus libre. Un exemple détaillé de ce prompt est disponible en Annexe A.2.

Ces deux méthodologies permettent d'analyser les performances des modèles sous différents angles, notamment en termes de précision, de cohérence des résultats et de conformité aux exigences du projet, en particulier pour l'intégration des résultats dans le système de gestion des tickets.

7.2 Expérimentations choix des modèles et combinaisons testés

Après avoir comparé plusieurs modèles de 13 milliards de paramètres, nous avons retenu les combinaisons suivantes de plateformes, de prompts et de modèles pour nos expérimentations.

Sur la plateforme Ollama, avec un prompt au format JSON, nous avons testé

— Llama3.2-11B-Vision, Qwen2.5-14B et Mistral-Nemo-12B.

En parallèle, sur la plateforme Unsloth, avec un prompt en langage naturel, nous avons évalué

— Llama3.2-11B-Vision, Qwen2.5-14B et Pixtral-12B.

Deux modèles, Llama3.2-11B-Vision et Qwen2.5-14B, ont été déployés sur les deux plateformes, permettant ainsi d'analyser leurs performances dans des contextes distincts et d'évaluer leur capacité d'adaptation à différentes structures de prompts.

7.3 Vérification et normalisation des sorties générées en langage naturel et au format JSON

Dans le cadre de la génération de sorties en langage naturel ou au format JSON, il est fondamental d'évaluer si les prédictions correspondent aux catégories attendues. Cette vérification repose sur un ensemble de critères normalisés permettant d'assurer la cohérence et la fiabilité des résultats.

7.3.1 Définition d'une correspondance valide

Une prédiction est considérée comme *matched* si elle peut être associée de manière univoque à l'une des catégories définies dans notre référentiel. Cette correspondance est établie en fonction de critères linguistiques et statistiques.

7.3.2 Normalisation des prédictions

Afin de réduire les variations introduites par des erreurs de formatage ou de syntaxe, un processus de normalisation est appliqué aux prédictions. Celui-ci comprend :

— Suppression des caractères spéciaux, afin d'éliminer les éléments non significatifs pour la comparaison.

— Conversion en minuscules, permettant d'uniformiser les entrées et de minimiser les biais liés à la casse.

7.3.3 Méthodes d'évaluation des correspondances

L'association entre une prédiction et une catégorie attendue repose sur une combinaison de mesures textuelles permettant d'évaluer la similarité sémantique et syntaxique :

— *Distance de Levenshtein* [11] : Quantification du nombre minimal d'opérations (insertion, suppression, substitution) nécessaires pour transformer une chaîne de caractères en une autre.

— Ratio de similarité de séquences : Application de l'algorithme *SequenceMatcher* [4] pour mesurer le degré de similitude entre les chaînes textuelles.

— Ratio de correspondance des mots (*word_match_ratio*) : Comparaison basée sur la proportion de mots communs entre la prédiction et la catégorie de référence.

7.3.4 Seuils d'acceptation des correspondances

Pour qu'une prédiction soit considérée comme valide, elle doit satisfaire les critères suivants :

— $DistanceLevenshtein \leq 3$, garantissant une proximité lexicale acceptable.

— $RatioSimilarité \geq 85\%$, reflétant un degré élevé de correspondance textuelle.

— $RatioCorrespondanceMots \geq 80\%$, assurant une cohérence terminologique suffisante.

Cette approche permet d'optimiser la robustesse de la catégorisation automatique et d'améliorer la fiabilité des prédictions en minimisant les erreurs de classification.

7.4 Résultats des expérimentations

Les résultats obtenus sont les suivants :

Mod.	Plat.	Mat. Pred.	Unmat. Pred.	Match Rate	Acc.
Llama	Olla.	891	1	0,9988	0,8049
Qwen	Olla.	850	42	0,9529	**0,8520**
Mistral-Nemo	Olla.	659	233	0,7645	0,6446
Pixtral	Unsl.	465	427	0,5213	0,4159
Qwen	Unsl.	523	369	0,5863	0,4776
Llama	Unsl.	746	146	0,8363	0,4540

TABLE 2 – Benchmark des modèles initiaux (Mod. : Modèle, Plat. : Plateforme, Mat. Pred. : Matched Predictions, Unmat. Pred. : Unmatched Predictions, Match Rate : Matching Rate, Acc. : Accuracy (exactitude), Olla. : Ollama (prompt JSON), Unsl. : Unsloth (prompt en langage naturel)

Voici une explication détaillée des concepts :

— Matched Predictions : Cela correspond aux prédictions du LLM qui génèrent un nom de catégorie valide (parmi les catégories attendues).

— Unmatched Predictions : Ces prédictions correspondent à des résultats où le modèle génère une réponse qui n'appartient pas à l'ensemble des catégories pré-définies.

— Matching Rate :

$$\frac{MatchedPredictions}{MatchedPredictions + UnmatchedPredictions}$$

— Accuracy (exactitude) : l'exactitude mesure le pourcentage de prédictions correctes.

7.5 Analyse des résultats

Les modèles utilisant Ollama et le prompt JSON présentent des taux de correspondance (matching rate) et des exactitudes nettement supérieurs, en particulier pour Llama3.2-11B-Vision et Qwen2.5-14B.

Les performances des modèles sur la plateforme Unsloth, en particulier avec le prompt en langage naturel, sont globalement plus faibles, ce qui peut être attribué à l'absence de structure formelle dans les résultats générés. Pour une analyse plus approfondie des performances, nous avons généré des matrices de confusion pour les modèles Ollama-Llama3.2-11B-Vision et Ollama-Qwen2.5-14B.

Ollama-Llama3.2-11B-Vision :

— Le modèle excelle dans les catégories majoritaires comme *incidents/supervision* (526 prédictions correctes).

FIGURE 1 – Matrice de confusion pour Ollama-Llama3.2-Vision

— Cependant, des confusions subsistent dans les catégories moins fréquentes telles que *demande de service/backup bcs/stratégie de sauvegarde/suppression*, avec une faible exactitude.
— La catégorie *UNMAPPED* correspond à « impossible à classer ». Il y a deux cas distincts. Premièrement, conformément aux consignes du prompt, le modèle retourne *UNKNOWN* en cas d'incertitude sur la classification, ce résultat étant affiché comme *UNMAPPED* dans la figure. Deuxièmement, le LLM génère des catégories totalement impossibles à identifier, même après vérification et normalisation, comme mentionné précédemment.

FIGURE 2 – Matrice de confusion pour Ollama-qwen2.5

Ollama-Qwen2.5-14B :
— Avec une exactitude globale de 85,20 %, ce modèle se distingue par sa gestion efficace de catégories comme demande de *service/backup bcs/stratégie de sauvegarde/suppression*, où il surpasse Llama3.2 en termes de classifications correctes (23 contre 14).
— Bien que ses performances globales soient élevées, le taux d'erreurs dans *UNMAPPED* est légèrement supérieur à celui de Llama3.2.

Ces expérimentations mettent en lumière l'importance d'une ingénierie des prompts adaptée et d'une plateforme robuste pour maximiser les performances des modèles. Les résultats obtenus serviront de base pour les étapes ultérieures d'optimisation et de fine-tuning.

8 Procédure d'entraînement (Fine-Tuning)

Dans cette section, nous décrivons le processus de fine-tuning appliqué aux modèles Llama 3.2 Vision 11B et Qwen 2.5 14B, en mettant en avant les motivations, les configurations techniques, et les résultats préliminaires.

8.1 Contexte et sélection des modèles

Sur la base des résultats des expérimentations initiales, deux modèles ont été retenus pour le fine-tuning :
Llama 3.2 Vision 11B : Modèle multimodal performant, ayant démontré une exactitude initiale élevée dans les tâches de classification.
Qwen 2.5 14B : Modèle purement linguistique, reconnu pour ses capacités à gérer les tâches de compréhension et d'extraction dans des contextes complexes.
Le choix de ces modèles repose principalement sur deux critères essentiels. Tout d'abord, leur performance initiale s'est révélée nettement supérieure lors des évaluations comparatives, attestant de leur efficacité et de leur pertinence pour les tâches ciblées.
Ensuite, la question de la compatibilité avec les plateformes a également joué un rôle déterminant. En effet, bien que le modèle Llama 3.2 Vision 11B présente des capacités multimodales avancées, les contraintes actuelles de la plateforme Ollama en matière d'intégration de modèles multimodaux privés compliquent son déploiement [1] [2]. Cette limitation souligne l'importance d'optimiser Qwen 2.5 14B, un modèle exclusivement textuel, afin d'assurer une intégration plus fluide et efficiente au sein de l'environnement ciblé.

8.2 Configurations techniques du fine-tuning

Dans notre démarche d'optimisation du fine-tuning, nous avons adopté la plateforme Unsloth, privilégiant une approche de quantification du modèle afin d'améliorer l'efficacité en termes de consommation mémoire et de rapidité de convergence. L'utilisation de modèles pris en charge par Unsloth permet ainsi d'exploiter pleinement les ressources matérielles disponibles, tout en garantissant des performances optimales.
Tous nos fine-tuning sont réalisés à partir de la version quantifiée (4 bits) des modèles d'origine, ce qui réduit significativement l'empreinte mémoire et permet l'entraînement sur des GPU disposant d'une VRAM limitée.
Dans notre cas spécifique, la tâche est exclusivement linguistique et ne nécessite pas de traitement d'images. Toute-

1. https ://github.com/unslothai/unsloth/issues/1504
2. https ://github.com/ollama/ollama/issues/7912

fois, Llama 3.2 Vision 11B étant un modèle multimodal, nous avons mis en place des mesures visant à préserver ses capacités en vision dans l'éventualité d'un usage futur, comme l'intégration du traitement d'images jointes aux e-mails. Pour ce faire, nous figeons les poids de la partie « image » (*image tower*) et ciblons le fine-tuning uniquement sur les couches linéaires du modèle de langage, en excluant notamment la *lm_head* afin de préserver la stabilité du modèle [5, 2]. Concrètement, nous nous concentrons sur l'ajustement

- des couches d'attention (*q_proj*, *k_proj*, *v_proj*, *o_proj*)
- ainsi que des couches de MLP (*gate_proj*, *up_proj*, *down_proj*).

Pour le modèle Qwen 2.5 14B, nous appliquons un entraînement sur l'ensemble des couches linéaires, à l'exception de la *lm_head*.

Au cours de nos expériences, nous avons testé plusieurs configurations de paramètres afin d'évaluer leur impact sur les performances du modèle. Dans cet article, nous présentons les résultats d'entraînement obtenus à partir d'un ensemble de paramètres optimisés. Ce choix vise à équilibrer l'utilisation des ressources computationnelles et la performance du modèle, garantissant ainsi un entraînement stable et efficace, même dans un environnement matériel contraint.

Pour garantir un benchmark équitable, nous avons utilisé les mêmes configurations pour le fine-tuning des deux modèles : Llama 3.2 Vision 11B et Qwen 2.5 14B. Ci-dessous, la configuration spécifique des modèles fine-tunés :

- Paramètres PEFT (Parameter Efficient Fine-Tuning)
  - *r=16* : Rank élevé pour garantir une précision optimale.
  - *lora_alpha=16* : Aligné avec la valeur de r.
  - *lora_dropout=0* : Aucun dropout appliqué.
  - *bias="none"* : Pas de biais utilisé.
  - *use_rslora=True* : Stabilise et optimise l'entraînement.
  - *loftq_config=None* : LoftQ non activé.
- Paramètres d'entraînement :
  - Batch size par appareil : 2.
  - Gradient accumulation steps : 4.
  - Étapes de warmup : 5.
  - Taux d'apprentissage : 2e-4.
  - Précision : bf16.
  - Optimiseur : *adamw_8bit*.
  - Décay : 0,01.
  - Longueur maximale des séquences : 2048.
  - Quantification : 4 bits.

8.3 Expérimentations des fine-tunings

Pour chaque modèle, deux versions ont été entraînées : une avec 30 steps de mise à jour des poids, et une autre avec 1 époque complète (correspondant à environ 440 steps). Ces versions permettent d'évaluer l'impact du nombre d'itérations d'entraînement sur les performances des modèles.

Pour analyser les performances des modèles, nous avons examiné l'évolution de la perte (loss) au cours de l'entraînement. L'évaluation s'est concentrée sur la stabilité et la convergence de la perte dans le cadre d'une époque complète (1ep).

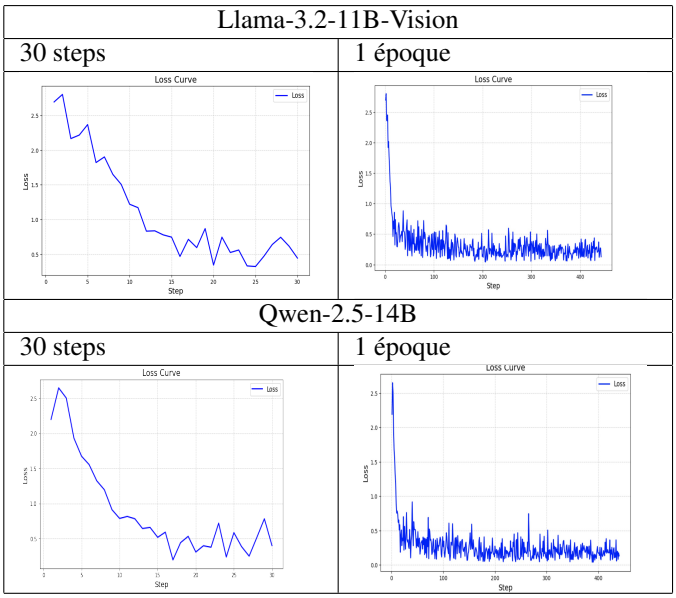

TABLE 3 – Courbes de perte (Loss curves) des fine-tunings

Modèle Qwen-2.5-14B : Ce modèle a montré une perte moyenne de 0,8505 avec un écart-type de 0,6461 sur les 30 dernières étapes. Ces résultats suggèrent une bonne capacité de convergence avec une perte relativement faible et des variations limitées, témoignant d'une grande stabilité en fin d'entraînement. De plus, la valeur minimale de perte atteinte, 0,1962, illustre son potentiel à fournir des prédictions précises.

Modèle LLaMA-3.2-11B-Vision : En comparaison, LLaMA a affiché une perte moyenne légèrement plus élevée de 1,0489, accompagnée d'un écart-type de 0,6963. Les variations de la perte, bien que modérées, sont légèrement supérieures à celles de Qwen. La valeur minimale de perte atteinte, 0,3238, bien qu'encourageante, reste au-dessus de celle observée pour Qwen.

En conclusion, sur le critère de la perte, le modèle Qwen surpasse LLaMA, non seulement par une perte moyenne inférieure, mais également par une meilleure stabilité en fin d'entraînement. Ces résultats confirment que Qwen est mieux adapté à converger rapidement tout en minimisant les erreurs.

9 Benchmark des modèles fine-tunés

Cette section présente les performances des modèles fine-tunés sur deux plateformes (Ollama et Unsloth) et selon deux formats de sortie (JSON imposé et texte libre simulant du JSON). Au total, six variantes ont été évaluées :

- *Qwen-Ollama-JSON* : 1 époque et 30 steps
- *Qwen-Unsloth–"naturel"* : 1 époque et 30 steps
- *Llama-Unsloth–"naturel"* : 1 époque et 30 steps

Sur Ollama, la sortie est systématiquement contrainte au format JSON. En revanche, le modèle Llama fine-tuné,

étant multimodal, présente des problèmes de compatibilité avec cette plateforme (voir Section 8.1).
Concernant Unsloth, bien que les sorties soient qualifiées de « naturelles », le prompt oriente tout de même le modèle vers une structure JSON (ex. *Output : {'categorisation' : ...}*). Un post-traitement similaire à celui décrit en Section 7.3 est ensuite appliqué pour harmoniser les résultats.

9.1 Résultats des expérimentations des modèles fine-tunés

Mod.	Plat.	Mat. Pred.	Unmat. Pred.	Match Rate	Acc.
Qwen 1ep.	Olla.	891	1	0,9989	0,8285
Qwen 30steps	Olla.	873	19	0,9787	0,8509
Qwen 1ep.	Unsl.	**892**	**0**	**1,0**	**0,8812**
Qwen 30steps	Unsl.	886	6	0,9933	0,8565
Llama 1ep.	Unsl.	889	3	0,9966	0,8610
Llama 30steps	Unsl.	870	22	0,9753	0,8105

TABLE 4 – Benchmark des modèles fine-tunés (Mod. : Modèle, Plat. : Plateforme, Mat. Pred. : Matched Predictions, Unmat. Pred. : Unmatched Predictions, Match Rate : Matching Rate, Acc. : Accuracy (exactitude), Olla. : Ollama (prompt JSON), Unsl. : Unsloth (prompt en langage naturel)

9.2 Analyse des performances des modèles fine-tunés

Les modèles Qwen et LLaMA montrent une amélioration significative de leurs performances après fine-tuning. Par exemple, LLaMA passe d'une exactitude initiale de 0,4540 (avant micro-ajustement) à plus de 0,8610, surpassant ainsi le score de 0,8049 obtenu avec la version non ajustée sous Ollama.
Le meilleur score global (0,8812) est obtenu avec Qwen (1 époque) sur Unsloth, en cohérence avec la tendance observée dans l'évolution de la fonction de perte (loss).

9.3 Analyse des matrices de confusion

Une analyse des matrices de confusion des modèles fine-tunés (1 époque) sur Unsloth révèle les observations suivantes :

Qwen - 1 époque - Unsloth (sortie naturelle)

- La majorité des classes sont correctement identifiées, en particulier les catégories majoritaires telles que *Incidents/Supervision* ou *Incidents/Backup BCS/Sauvegarde*.
- Les catégories moins fréquentes (ex. *Demande de service/Backup #BCS/Stratégie de sauvegarde/Suppression*) sont également bien prédites,

FIGURE 3 – Matrice de confusion pour Unsloth-qwen2.5 affiné en 1 époque

avec un faible taux de confusion.

FIGURE 4 – Matrice de confusion pour Unsloth-Llama3.2-Vision affiné en 1 époque

LLaMA - 1 époque - Unsloth (sortie naturelle)

- Une amélioration significative est observée par rapport à la version non ajustée, y compris pour les classes minoritaires, grâce au fine-tuning.
- Les confusions résiduelles concernent principalement des catégories proches, mais restent limitées compte tenu de la taille du jeu de données.

Dans l'ensemble, les fines distinctions entre catégories proches sont mieux gérées lorsque le modèle est spécifiquement entraîné sur le jeu de données annoté. Le fine-tuning permet ainsi d'affiner la capacité de classification, de réduire le taux de prédictions *UNMAPPED* et d'augmenter l'exactitude globale.

9.4 Discussion sur l'écart de performance sous Ollama

Les légers reculs observés sur Ollama peuvent être attribués à plusieurs facteurs. Tout d'abord, la conversion et la quan-

tification du modèle vers le format *gguf* introduisent des approximations supplémentaires liées à la re-quantification et à la compression, ce qui peut altérer légèrement la précision. De plus, des différences dans les pipelines d'entraînement et d'exécution pourraient également expliquer cette baisse de performance. Le fine-tuning ayant été réalisé via Unsloth, le modèle a ensuite été porté sous Ollama, un processus impliquant plusieurs transformations susceptibles d'affecter la cohérence des poids finaux.

10 Conclusion et perspectives

10.1 Conclusion

Dans le cadre de ce travail, nous avons évalué plusieurs modèles de langage d'environ 13 milliards de paramètres dans un environnement industriel, démontrant qu'il est possible d'atteindre un niveau de performance élevé (une exactitude de 88,12%) tout en respectant des contraintes matérielles et de confidentialité.

L'approche multi-agents retenue, qui sépare la classification et l'extraction d'informations en différents « agents », a été mise en œuvre dans le cadre de ce projet. Nous pensons qu'elle offre une modularité propice à de futures évolutions et un gain de maintenance, mais une étude plus approfondie reste à mener pour valider formellement son efficacité à grande échelle.

De plus, nos expérimentations soulignent l'importance de la quantification (en 4 bits) et du *fine-tuning* ciblé (via PEFT/LoRA), solutions qui permettent de concilier efficacité opérationnelle et qualité des prédictions dans un contexte de ressources limitées.

L'ensemble des travaux présentés dans cet article, incluant le fine-tuning et les évaluations comparatives, est mis à disposition en accès libre sur le dépôt GitHub[3], à l'exception des jeux de données soumis à des contraintes de confidentialité.

10.2 Perspectives

À court terme, les perspectives s'organisent principalement autour de deux axes :

1. Améliorations de l'agent de classification.

Bien que la première version de l'agent de classification ait démontré une performance prometteuse, des enrichissements restent prévus : l'équipe projette d'élargir et d'actualiser le jeu de données d'entraînement afin d'affiner à nouveau le modèle Qwen2.5. L'objectif est de mieux couvrir l'éventail des catégories de tickets et de consolider la robustesse du système dans des scénarios réels encore plus variés.

2. Mise en place des agents d'extraction.

Le développement d'agents spécialisés dans l'extraction d'informations à partir des e-mails clients représente la prochaine étape clé du projet. Une première preuve de concept a été réalisée sur la catégorie « Restauration qualifiée », nécessitant l'identification de cinq champs : la date de restauration, le chemin source, le type, le chemin de destination, le nom d'hôte. Nous avons déjà fait les benchmarks des modèles initiaux pour cette tâche. Les résultats indiquent que Qwen 2.5 14B et Llama 3.2 13B Vision restent deux candidats de choix pour cette tâche, mais la constitution d'un corpus annoté demeure un défi. L'équipe prévoit donc d'exploiter une stratégie d'auto-annotation [14], puis de valider un échantillon des données par des experts humains avant de procéder au fine-tuning final. Cette approche devrait accélérer le processus d'étiquetage, tout en garantissant la qualité des annotations indispensables à l'entraînement d'un modèle d'extraction fiable.

À long terme, notre ambition est d'intégrer l'ensemble de ces agents (classification et extraction) dans une architecture unifiée, afin de faciliter la génération automatique de tickets et leur suivi complet. Enrichir et diversifier les données, affiner les prompts et perfectionner les algorithmes de fine-tuning feront partie des travaux futurs. Ainsi, nous espérons rendre ces technologies d'intelligence artificielle de plus en plus accessibles et performantes pour les besoins industriels de traitement d'e-mails et de support client.

3. https://github.com/LeviatanAI/email-ticket-classifier

Références

[1] Michael Han Daniel Han and Unsloth team. Unsloth, 2023.

[2] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora : Efficient finetuning of quantized llms, 2023.

[3] Hugging Face. Open llm leaderboard. Accessed : 2025-02-21.

[4] Python Software Foundation. *difflib — Helpers for computing deltas*, 2025. Accessed : 2025-02-21.

[5] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora : Low-rank adaptation of large language models, 2021.

[6] Shiyao Li, Xuefei Ning, Luning Wang, Tengxuan Liu, Xiangsheng Shi, Shengen Yan, Guohao Dai, Huazhong Yang, and Yu Wang. Evaluating quantized large language models, 2024.

[7] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. Peft : State-of-the-art parameter-efficient fine-tuning methods, 2022.

[8] Meta. Llama 3.2 : Revolutionizing edge ai and vision with open, customizable models, September 2024. Accessed : 2025-02-21.

[9] Ollama. Ollama. Accessed : 2025-02-21.

[10] OpenAI. Openai platform models. Accessed : 2025-02-21.

[11] Wikipedia contributors. Distance de levenshtein — wikipedia, the free encyclopedia, 2025. Accessed : 2025-02-21.

[12] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian

Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2024.

[13] Yingxuan Yang, Qiuying Peng, Jun Wang, Ying Wen, and Weinan Zhang. Llm-based multi-agent systems : Techniques and business perspectives, 2024.

[14] Ying Zhang, Matthieu Petit Guillaume, Aurélien Krauth, and Manel Labidi. Cryptogpt : a 7b model rivaling gpt-4 in the task of analyzing and classifying real-time financial news, 2024.

A Annexe

A.1 Prompt avec sortie en format JSON sous Ollama

```
Tu es un assistant spécialisé dans la
classification de tickets à partir de
leur contenu textuel. Ton objectif est
d'analyser la description fournie et
de l'associer à l'une des catégories
ci-dessous:
```
- Demande de service/Backup #BCS/Autre
- Demande de service/Backup
#BCS/Demande de renseignement
- Demande de service/Backup
#BCS/Restauration qualifiée
- Demande de service/Backup
#BCS/Stratégie de sauvegarde/Création
- Demande de service/Backup
#BCS/Stratégie de sauvegarde/Modification
- Demande de service/Backup
#BCS/Stratégie de sauvegarde/Suppression
- Demande de service/Cyber Sécurité
#CS2/Bastion/Création-Modification
d'entrées
- Incidents/Backup #BCS/Sauvegarde
- Incidents/Supervision
```
### Règles :
1. Réponds uniquement par la catégorie
exacte, sans texte supplémentaire.
2. La catégorie associée doit être
unique.
3. Si aucune catégorie ne correspond,
la valeur de "categorisation" doit être
"unknown".
### Exemple :
La description:
Bonjour¤ Merci de relancer les
sauvegardes FULL¤ si ils ne sont pas
repassées en automatique. Ghislain
Envoyé de mon iPhone Début du message
transféré
### Sortie :
{{
"categorisation": "Incidents/Backup
#BCS/Sauvegarde "
}}
Analyse uniquement la description
suivante :
{content}
```

A.2 Prompt avec sortie en langage naturel sous Unsloth

```
Tu es un assistant spécialisé dans la
classification de tickets à partir de
leur contenu textuel. Ton objectif est
d'analyser la description fournie et
de l'associer à l'une des catégories
ci-dessous:
```
- Demande de service/Backup #BCS/Autre
- Demande de service/Backup
#BCS/Demande de renseignement
- Demande de service/Backup
#BCS/Restauration qualifiée
- Demande de service/Backup
#BCS/Stratégie de sauvegarde/Création
- Demande de service/Backup
#BCS/Stratégie de sauvegarde/Modification
Demande de service/Backup
#BCS/Stratégie de sauvegarde/Suppression
- Demande de service/Cyber Sécurité
#CS2/Bastion/Création-Modification
d'entrées
- Incidents/Backup #BCS/Sauvegarde
- Incidents/Supervision
"'
```
### Règles :
1. Réponds uniquement par la catégorie
exacte, sans texte supplémentaire, la
réponse doit être encapsuler par deux
*.
2. Une seule catégorie maximum sera
retenue.
3. Si aucune catégorie ne correspond,
réponds simplement par *unknown*.
### Exemple :
La description:
Bonjour¤ Merci de relancer les
sauvegardes FULL¤ si ils ne sont pas
repassées en automatique. Ghislain
Envoyé de mon iPhone Début du message
transféré
### Sortie :
*Incidents/Backup #BCS/Sauvegarde*
Analyse uniquement la description
suivante :
{content}
```