
Rapport Intermédiaire

FINE TUNING D'UN LLM POUR UN CAS D'USAGE
SPÉCIFIQUE

PROJET INFONUM 5

Mardi 19 Decembre 2023



Team members:
Ayman MOUMEN
Marouane MAAMAR
Samer LAHOUD

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 2 |
| 2 | Contexte et problématique : | 2 |
| 2.1 | Problématique initiale | 2 |
| 2.2 | Echange avec OnePoint : | 2 |
| 3 | Objectifs fixés et cas d'usages identifiés : | 3 |
| 3.1 | Objectifs fixés : | 3 |
| 3.2 | Livrables attendus | 3 |
| 3.3 | Contraintes du projet : | 3 |
| 3.4 | Cas d'usages identifiés : | 4 |
| 4 | Analyse des points clés : | 4 |
| 4.1 | Code LLMs : | 4 |
| 4.1.1 | Evaluation des solutions déployées actuellement : | 5 |
| 4.1.2 | Evaluation des performances des modèles OpenSource : | 5 |
| 4.2 | Méthodes de Fine tuning : | 6 |
| 4.2.1 | L'architecture des LLM (Transformers) | 6 |
| 4.2.2 | Fine-tuning des LLM | 7 |
| 4.2.3 | État de l'Art des Méthodes de Fine-Tuning | 8 |
| 4.2.4 | Méthodes de Fine-Tuning : LoRA et PEFT | 9 |
| 4.3 | Méthodes d'évaluation : | 11 |
| 4.3.1 | BLEU | 11 |
| 4.3.2 | ROUGE | 11 |
| 4.3.3 | HumaEval | 12 |
| 4.3.4 | Autres méthodes d'évaluation | 13 |
| 5 | Shéma d'ensemble et finalité prévue : | 14 |
| 5.1 | Shéma d'ensemble : | 14 |
| 5.2 | Diagramme de Gantt : | 15 |
| 6 | Etat d'avancement et résultats obtenus : | 15 |
| 6.1 | Fine tuning sur les langages de programmation : | 15 |
| 6.1.1 | Fine tuning sur un dataset de Finance : | 16 |
| 6.1.2 | Fine tuning sur Python : | 16 |
| 6.1.3 | Fine tuning sur un nouveau langage de programmation : | 16 |
| 7 | Conclusion et tâches futures : | 17 |
| 7.1 | Conclusion générale | 17 |
| 7.2 | Problèmes rencontrés : | 17 |
| 7.3 | Tâches futures : | 18 |
| 8 | Références : | 19 |

1 Introduction

Le Groupe OnePoint occupe une place prépondérante dans le paysage des technologies de l'information et des services numériques en France. Fondée sur une vision novatrice et une expertise approfondie, cette entreprise s'est érigée en un acteur majeur du conseil, contribuant de manière significative à la transformation digitale des entreprises. Au cœur de ses activités, OnePoint déploie des solutions innovantes et des stratégies sur mesure pour répondre aux défis complexes de l'écosystème numérique contemporain.

2 Contexte et problématique :

2.1 Problématique initiale :

Initialement, le projet consiste en un "**fine-tuning**" d'un modèle de langage LLM léger, tel que distilGPT, sur un ensemble de données spécifique. L'objectif est d'explorer différentes techniques de "fine-tuning" et d'évaluer les performances du modèle ainsi ajusté. Les compétences développées au cours du projet devraient inclure la compréhension théorique des LLM, l'utilisation de bibliothèques open source telles que PEFT, LoRa, et évaluation des performances des modèles obtenus.

Le projet se déroule à 50% en présentiel dans les locaux de l'entreprise à Paris, avec des points de suivi hebdomadaires et des présentations régulières aux parties prenantes, notamment les leaders et partenaires de One Point. Le but était de nous donner l'opportunité de monter en compétence grâce à l'accompagnement par des experts de One Point, tout en découvrant le monde du conseil. En fin de projet, nous devrons avoir créé un livrable démonstratif qui pourra être utilisé comme exemple concret pour les clients.

2.2 Echange avec OnePoint :

Afin de cibler efficacement le vaste projet de fine-tuning de LLM, il était impératif de définir un cas d'usage spécifique qui répondrait aux besoins et aux intérêts particuliers de OnePoint. Le défi résidait dans l'identification d'une problématique claire et pertinente, offrant ainsi une direction précise pour le projet.

Nous avons alors entrepris l'exploration de différents cas d'usage potentiels, en mettant l'accent sur la recherche d'une problématique engageante et susceptible d'interesser OnePoint.

Dans le cadre de l'analyse approfondie des opportunités de projet, nous avons entrepris une démarche structurée en classifiant les idées par thème, dans un fichier excel (**Lien du document**), allant de l'utilisation des modèles de LLM dans le secteur financier (tel que l'utilisation d'un LLM pour classer la pertinence des cryptomonnaies en se fondant sur l'analyse des opinions exprimées sur les réseaux sociaux) à leur application dans le domaine de l'énergie. Notre objectif était de cerner des projets potentiellement innovants et alignés sur les domaines d'intérêt de OnePoint.

3 Objectifs fixés et cas d'usages identifiés :

3.1 Objectifs fixés :

L'objectif principal de ce projet est d'approfondir la compréhension des techniques de fine-tuning appliquées aux modèles de génération de texte actuels, tout en évaluant leur pertinence et leurs performances pour OnePoint et ses clients potentiels. Pour ce faire, plusieurs sous-objectifs sont définis :

- **Exploration de l'état de l'art des méthodes de fine-tuning :** Examiner et comprendre les méthodes de fine-tuning existantes appliquées aux LLM légers, y compris les outils open source tels que PEFT, LoRa et QLora . afin de sélectionner les approches les plus adaptées au contexte du projet.
- **Analyse des modèles de génération de texte actuels :** Étudier différents modèles de génération de texte disponibles, évaluer leurs performances initiales et identifier ceux qui pourraient bénéficier le plus du fine-tuning pour répondre aux besoins spécifiques de OnePoint.
- **Exploration de plusieurs modèles et datasets :** Expérimenter avec divers ensembles de données et modèles de LLM, en évaluant leur pertinence et leur efficacité potentielle pour les besoins de fine-tuning de OnePoint et de ses clients.
- **Comparaison des méthodes de fine-tuning :** Comparer différentes méthodes de fine-tuning en termes de ressources nécessaires, temps requis et performances des modèles après fine-tuning.

3.2 Livrables attendus

Un repo GitHub détaillé contenant les différents fichiers de code et notebooks utilisés pour le fine-tuning des modèles. Chaque étape du processus de fine-tuning sera expliquée en détail, y compris la méthodologie utilisée, le choix des paramètres et une évaluation approfondie des résultats obtenus, permettant ainsi d'offrir un exemple concret et utilisable pour les clients de OnePoint.

3.3 Contraintes du projet :

Suite aux réunions avec le tuteur plusieurs contraintes ont été établies pour guider notre avancement dans le cadre du projet :

1. **Disponibilité et Utilisation des Ressources :** Un point essentiel discuté était la nécessité de planifier et de réserver en avance les ressources de l'environnement Mydocke doté d'un GPU puissant, fournies par l'école, pour garantir un accès fluide aux capacités de calcul et de stockage requises.
2. **Documentation Méthodologique :** La nécessité de documenter minutieusement chaque étape du processus de fine-tuning dans des notebooks a été soulignée. Cette documentation détaillée garantira la clarté et la reproductibilité des démarches entreprises.
3. **Critères de Sélection des Modèles et datasets:** La nécessité de documenter minutieusement chaque étape du processus de fine-tuning dans des notebooks a été soulignée. Cette documentation détaillée garantira la clarté et la reproductibilité des démarches entreprises ainsi qu'une claire comparaison des différentes méthodes de fine tuning utilisées ainsi que les datasets utilisés pour entraîner le modèle.

4. **Communication et Présentation des Résultats** : La communication régulière avec le tuteur du projet et la préparation de présentations claires et concises pour les parties prenantes dès l'obtention de résultats significatifs ont été définies comme des impératifs pour assurer la validation continue et la transparence du projet.

3.4 Cas d'usages identifiés :

Suite à nos échanges avec OnePoint, Thomas, notre encadrant de projet s'est montré particulièrement enthousiaste vis à vis des projets liés à la digitalisation et à la programmation. Au travers de cette exploration, deux concepts ont émergé comme étant particulièrement pertinents pour les besoins de One Point.

- Le premier porte sur l'exploitation des LLM pour le code, avec un accent particulier sur le débogage et la résolution d'erreurs. Cette approche promet d'optimiser significativement les processus de développement et de maintenance des codes, présentant ainsi des avantages tangibles pour les équipes travaillant sur des projets de digitalisation.
- La deuxième idée majeure se concentre sur le fine-tuning de modèles de LLM spécifiquement pour le langage SAS. Cette proposition découle des défis de migration de code SAS rencontrés par One Point dans un contexte client. En adaptant les LLM aux particularités du langage SAS, nous visons à surmonter les obstacles actuels et à améliorer la compatibilité entre différentes versions de code, facilitant ainsi la transition pour le client.

La section 'Analyse des points clés' justifiera en détail nos choix pour ces cas d'usages.

4 Analyse des points clés :

4.1 Code LLMs :

- **Qu'est ce qu'un CODE LLM ?**

Un code LLM est utilisé pour générer du code et aider à la programmation. Il peut comprendre et produire du code dans plusieurs langages de programmation, aidant les développeurs dans des tâches telles que la complétion de code, le débogage, la rédaction de documentation et même la génération de code basé sur des descriptions en langage naturel. Le code LLM est principalement entraîné sur un dataset de codes sur différents langages de programmation. Ce genre de modèles est formé sur de vastes ensembles de données et peuvent générer des extraits de code ou des programmes complets basés sur des instructions de saisie. Ils constituent une avancée significative dans le domaine du développement de logiciels, permettant aux programmeurs de travailler plus facilement et plus efficacement sur des projets complexes et réduisant les erreurs de codage.

4.1.1 Evaluation des solutions déployées actuellement :

Afin de justifier les cas d'usages identifiés et leurs pertinences pour le groupe OnePoint, nous avions effectué un état de l'art des solutions existantes en soulignant leurs avantages et leurs limitations. Il existe plusieurs interfaces déployées actuellement qui assistent les développeurs et programmeurs dans leurs projets, nous avons décidé de résumer les avantages et limitations des différentes interfaces dans le schéma ci-dessous :

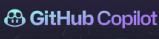
| |  GitHub Copilot |  Bard |  CodeWhisperer |  OpenAI code.interpreter() |
|-------------|--|---|--|---|
| Avantages | <ul style="list-style-type: none"> • Complète le code basé sur le contexte donné • Prise en charge de plusieurs langages | <ul style="list-style-type: none"> • Génère un code basé sur les instructions de l'utilisateur • Prise en charge de plusieurs langages • Explique les erreurs rencontrées • Dispose d'une interface web pour tout utilisateur | <ul style="list-style-type: none"> • Complète le code basé sur le contexte donné • Prise en charge de plusieurs langages | <ul style="list-style-type: none"> • Complète le code basé sur le contexte donné • Explique les erreurs et effectue le "débogage" • Dispose d'une interface web pour tout utilisateur • Prise en charge de plusieurs langages |
| Limitations | <ul style="list-style-type: none"> • Besoin d'être intégré dans un IDE (pas pratique pour LINUX/UNIX) • Ne peut pas déboguer ou expliquer les erreurs • Quelques problèmes d'hallucinations et de codes mal générés | <ul style="list-style-type: none"> • Faible face aux demandes d'algorithmes complexes • Beaucoup de problèmes d'hallucinations et de codes mal générés • Version préliminaire donc faible performance | <ul style="list-style-type: none"> • Besoin d'être intégré dans un IDE (pas pratique pour LINUX/UNIX) • Ne peut pas déboguer ou expliquer les erreurs • Quelques problèmes d'hallucinations et de codes mal générés • Faible face aux demandes d'algorithmes complexes | <ul style="list-style-type: none"> • Quelques problèmes d'hallucinations et de codes mal générés • Exclusif aux membres GPT+ • Rumeurs d'infractions de la confidentialité des données partagées |

Figure 2: Avantages et limitations des interfaces assistantes dans le développement de code

Il existe bien sûr d'autres interfaces qui ont été déployées pour assister dans le développement de codes, nous avons choisi de comparer les interfaces les plus utilisées et les plus connues.

4.1.2 Evaluation des performances des modèles OpenSource :

Nous souhaitons rappeler dans cette section que l'une des contraintes auquel nous devions faire face est d'utiliser des modèles OpenSource pour nos cas d'usage. Durant nos recherches nous avons quelques modèles code LLMs évalués sur HumanEval et MBPP(des métriques d'évaluation pour juger la qualité et la justesse des codes python générés, la section 'Méthodes d'évaluation' abordera ça en détail) et nous avons pu conclure les résultats suivantes :

| Model | HumanEval | MBPP |
|---------------------|-------------|-------------|
| LLaMA-7B | 10.5 | 17.7 |
| LaMDA-137B | 14.0 | 14.8 |
| LLaMA-13B | 15.8 | 22.0 |
| CodeGen-16B-Multi | 18.3 | 20.9 |
| LLaMA-33B | 21.7 | 30.2 |
| CodeGeeX | 22.9 | 24.4 |
| LLaMA-65B | 23.7 | 37.7 |
| PaLM-540B | 26.2 | 36.8 |
| CodeGen-16B-Mono | 29.3 | 35.3 |
| StarCoderBase | 30.4 | 49.0 |
| code-cushman-001 | 33.5 | 45.9 |
| Mistral-7B-Instruct | 32.1 | 47.5 |
| StarCoder | 33.6 | 52.7 |
| StarCoder-Prompted | 40.8 | 49.5 |

Table 1: Comparaison des Modèles OpenSource et ClosedSource sur HumanEval et MBPP en pass@1 [6]

On remarque que StarCoder et StarCoder-Prompted se démarquent du lot avec des scores assez élevés, ce sont de bons modèles qu'on a décidé de considérer pour notre projet. Toutefois, en raison du grand nombre de paramètres des modèles et de leur taille conséquente et vu l'environnement dont nous disposons (TESLA T4 16GB VRAM) sur Google Colab (avant qu'on y est accès à l'environnement MyDocker avec une Tesla V100 32GB VRAM), nous avons décidé de commencer des premiers tests avec Mistral-7B-Instruct qui a réussi à avoir des scores assez proches de StarCoder.

Un autre aspect à souligner, pour mettre en évidence l'intérêt du fine-tuning dans le débogage et l'assistance à la résolution d'erreurs, est que le modèle Starcoder a été entraîné uniquement sur l'auto-complétion et la génération de code, mais pas sur le débogage ni sur l'assistance à la résolution de problèmes. De plus, il n'a été fine-tuné que pour le langage Python, ce qui nous ouvre des possibilités de fine-tuning sur d'autres langages de programmation et de mesurer les performances correspondantes.

4.2 Méthodes de Fine tuning :

4.2.1 L'architecture des LLM (Transformers)

Les Transformers, en particulier les modèles de langage, reposent sur une architecture dénommée "décodeur" (Decoder) pour effectuer des tâches de génération de texte. Cette architecture est fondamentale pour les tâches de modélisation du langage naturel et a été révolutionnaire dans l'avancée des performances des modèles de traitement du langage.

Le schéma du décodeur Transformer illustre un réseau de neurones récurrents qui, contrairement aux modèles récurrents traditionnels, utilise une attention multi-têtes. Cette attention permet au modèle d'apprendre des dépendances longue-distance et d'exploiter des informations contextuelles à partir de l'ensemble de la séquence d'entrée pour générer des prédictions de mots suivants. Cette architecture se compose de plusieurs couches d'attention, de normalisation, et de réseaux de neurones entièrement connectés, favorisant ainsi l'apprentissage de représentations abstraites du langage.

L'architecture des modèles "decoder-only" comme Mistral se compose généralement des éléments suivants :

1. **Positional Encoding** : Introduit pour informer le modèle sur la position des mots ou des éléments dans la séquence. Cela permet au modèle de comprendre la séquence dans son contexte.
2. **Multi-Head Self-Attention Layers** : Ces couches permettent au modèle de comprendre les relations entre différents éléments de la séquence en les comparant les uns aux autres. Elles permettent une compréhension contextuelle des mots ou des symboles dans une séquence.
3. **Couches Feedforward** : Ces couches sont responsables de la transformation non linéaire des représentations apprises par les couches d'attention.
4. **Output Layers** : Ces couches finales génèrent les prédictions pour la séquence en sortie, que ce soit pour de la traduction, de la génération de texte ou d'autres tâches similaires.

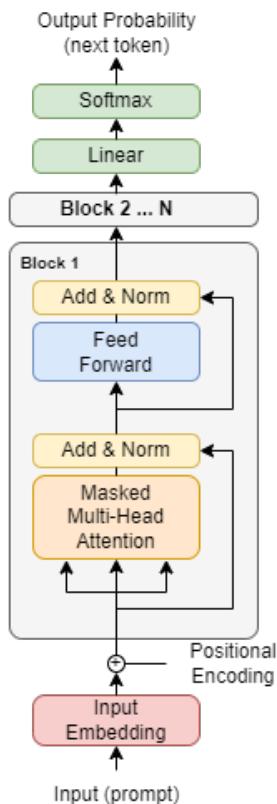


Figure 3: Architecture du transformer decoder

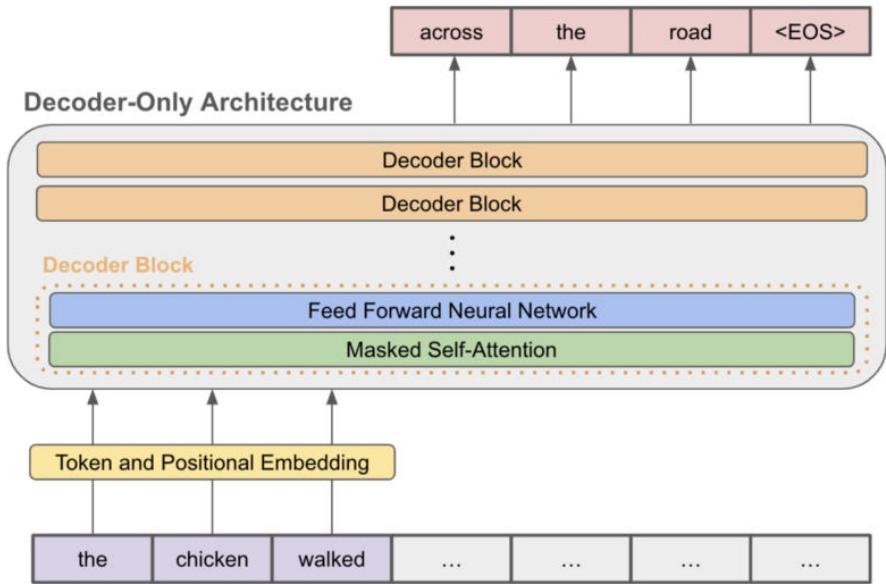


Figure 4: Architecture du transformer decoder-only

4.2.2 Fine-tuning des LLM

Le fine-tuning des LLM représente l'adaptation d'un modèle pré-entraîné à des données ou à des tâches spécifiques. Pour un modèle de langage, cette procédure implique généralement l'ajustement des paramètres du modèle afin de mieux s'aligner avec des données cibles plus spécifiques. Ces données peuvent concerner un domaine textuel particulier ou des types spécifiques de tâches linguistiques telles que la traduction, la génération de texte ou la compréhension de texte.

Ce processus de fine-tuning offre la possibilité d'optimiser les performances d'un modèle pour des tâches spécifiques sans exiger un entraînement complet à partir de zéro. En ajustant minutieusement les poids pré-entraînés du modèle, cette technique permet une adaptation ciblée, favorisant ainsi des résultats améliorés et plus précis pour des contextes ou des objectifs linguistiques définis.

4.2.3 État de l'Art des Méthodes de Fine-Tuning

Le domaine du fine-tuning des modèles de langage a évolué vers une multitude de techniques innovantes, visant à améliorer les performances et l'adaptabilité des modèles pré-entraînés sur des tâches spécifiques. Parmi les avancées remarquables :

- **Adaptive Fine-Tuning** : Cette approche dynamise le processus de fine-tuning en adaptant les taux d'apprentissage pour différentes parties du modèle. Elle favorise ainsi un ajustement plus précis et efficace lors de l'entraînement sur des données spécifiques.
- **Few-Shot Learning** : Les modèles, tels que GPT-3, ont démontré une capacité étonnante à exécuter des tâches avec un très petit nombre d'exemples d'entraînement. Cette méthode repousse les limites traditionnelles du fine-tuning en exploitant des stratégies d'apprentissage à partir de très peu de données.
- **Knowledge Distillation** : Cette approche implique la compression de modèles pré-entraînés plus grands pour les adapter à des modèles plus petits et plus efficaces tout en conservant leurs performances. Cela les rend plus adaptés à des déploiements dans des environnements contraints en ressources.
- **Adapter Layers** : L'ajout de couches supplémentaires spécifiques à la tâche (adapter layers) permet de cibler des aspects particuliers d'une tâche sans altérer les couches pré-entraînées. Cela offre une flexibilité accrue dans l'adaptation des modèles à des tâches spécifiques.
- **Multitask Fine-Tuning** : Cette approche combine plusieurs tâches d'apprentissage pour améliorer la capacité du modèle à généraliser. Elle lui permet de mieux comprendre et synthétiser différentes structures linguistiques, renforçant ainsi sa polyvalence.
- **Learning Rate Schedules** : La variation des taux d'apprentissage tout au long du processus de fine-tuning peut grandement améliorer la convergence du modèle et sa capacité à apprendre des représentations plus précises, optimisant ainsi ses performances.

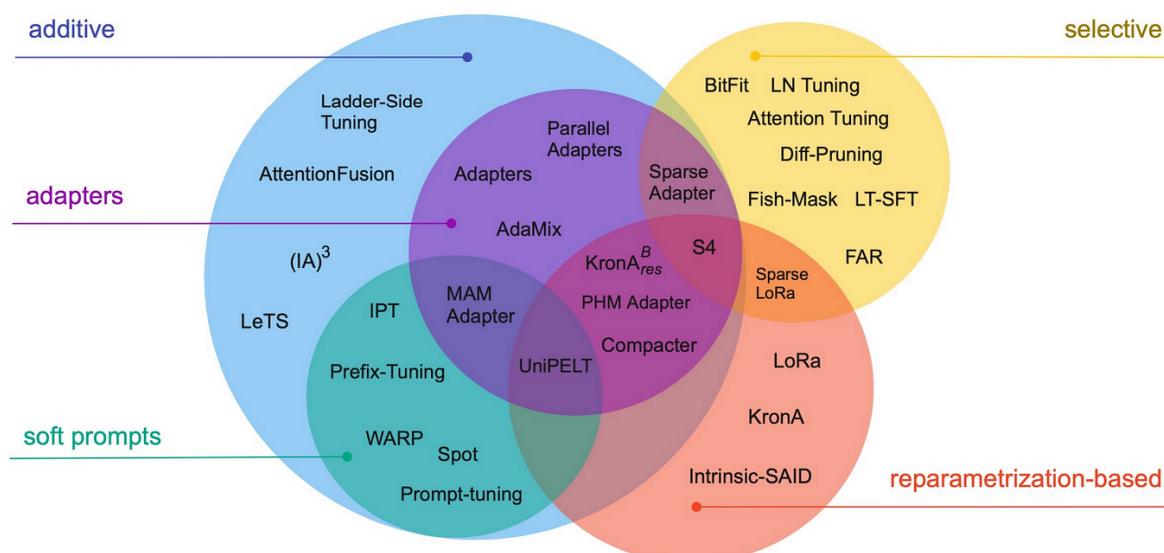


Figure 5: méthodes de PEFT

4.2.4 Méthodes de Fine-Tuning : LoRA et PEFT

Parmi les techniques de la méthode PEFT on trouve :

- **Adapters** : Les adaptateurs consistent à ajouter des modules ou des couches spécifiques à une tâche à un modèle pré-entraîné sans modifier de manière extensive ses paramètres initiaux. Ces adaptateurs permettent au modèle de se spécialiser pour une tâche spécifique tout en préservant les connaissances acquises lors de la pré-entraînement. Ils servent de modules compacts pouvant être insérés à différentes parties du réseau pour s'adapter à différentes tâches sans nécessiter de réentraînement approfondi.
- **Soft prompting** : Cette technique implique l'utilisation d'indications souples guidant l'attention du modèle vers certains aspects des données d'entrée. Au lieu d'indications rigides, les indications souples offrent une approche plus flexible en permettant au modèle de prêter doucement attention à des informations ou des indices spécifiques présents dans l'entrée, améliorant ainsi l'adaptabilité à différentes entrées.
- **Reparametrization** : La reparamétrisation modifie les paramètres du modèle lors du fine-tuning pour l'adapter à une tâche spécifique. Cela peut impliquer des modifications de certains aspects de l'architecture du modèle ou la modification de paramètres de manière spécifique à la tâche. Cela aide à adapter le comportement du modèle aux exigences de la tâche de fine-tuning sans altérer de manière extensive l'architecture initiale du modèle.

• LoRA (Low Rank Attention)

LoRA est une méthode de fine-tuning qui vise à réduire la complexité des calculs d'attention dans les modèles de langage. Plutôt que de calculer l'attention entre toutes les paires de mots, LoRA utilise une approximation de rang faible pour réduire la complexité temporelle et spatiale des opérations d'attention. Cela permet de réduire la charge computationnelle lors du fine-tuning, tout en préservant les performances du modèle.

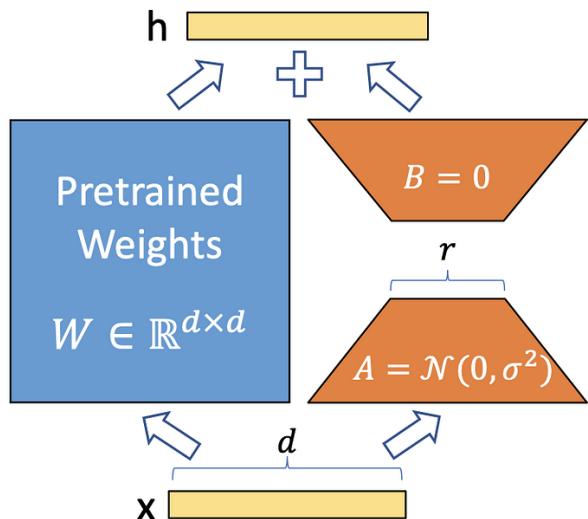


Figure 6: Lora 1

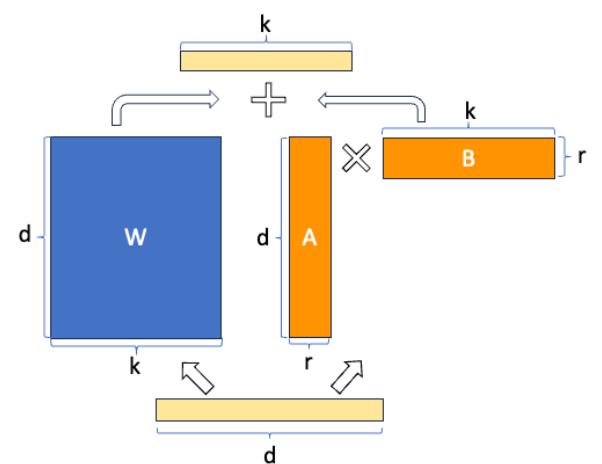


Figure 7: Lora 2

PEFT (Parameter Efficient Fine-Tuning)

PEFT se concentre sur l'optimisation des ressources lors du fine-tuning des modèles de langage. Cette méthode cherche à réduire la taille des modèles tout en maintenant leurs performances. Elle explore des techniques telles que la quantification de poids, la compression et la régularisation pour réduire le nombre de paramètres tout en préservant la capacité de généralisation du modèle.

l'équation de reparamétrisation :

$$W_0 + \Delta W = W_0 + BA \quad (1)$$

avec W_0 ($d \times k$), A ($d \times r$) et B ($r \times k$) et $r \ll d, k$

Parameter efficient fine-tuning (PEFT)

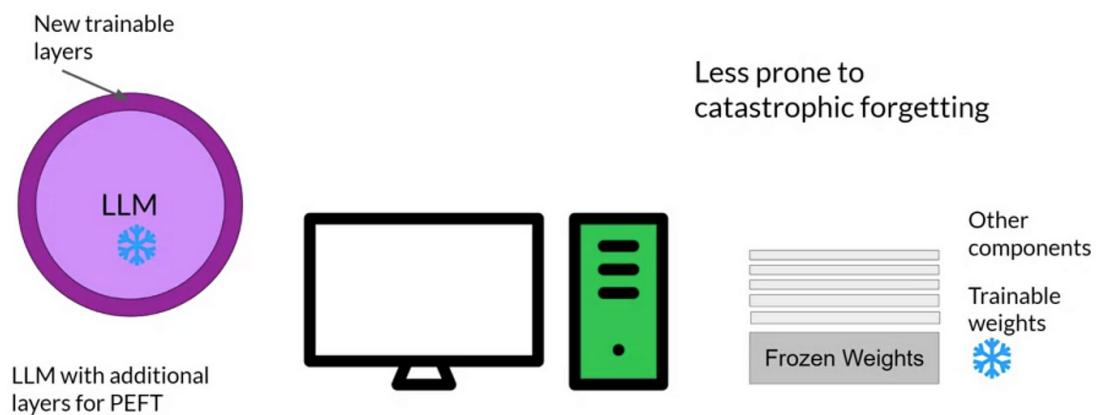


Figure 8: PEFT

Avantages et Applications

Ces méthodes, LoRA et PEFT, offrent des avantages significatifs pour le fine-tuning des modèles de langage, notamment en termes d'efficacité computationnelle, de réduction des ressources nécessaires et de maintien des performances du modèle. Elles sont particulièrement utiles dans des environnements où les contraintes de mémoire et de puissance de calcul sont importantes, tout en étant essentielles pour des déploiements à grande échelle.

Impact sur le Fine-Tuning

L'intégration de LoRA et PEFT dans le processus de fine-tuning ouvre la voie à des modèles plus efficaces et plus adaptés à une utilisation pratique. Ces techniques permettent d'exploiter les modèles de langage pré-entraînés tout en limitant la charge computationnelle et la consommation de ressources, ce qui les rend pertinents pour des applications variées, notamment dans les domaines nécessitant des modèles légers et rapides, comme le traitement de langage naturel sur des appareils mobiles ou des systèmes embarqués.

4.3 Méthodes d'évaluation :

L'évaluation des LLM constitue un domaine dynamique où diverses méthodes ont émergé pour mesurer la qualité et la pertinence de leurs sorties générées. Chaque méthode d'évaluation apporte son propre éclairage, offrant des perspectives distinctes sur la performance des LLM. Dans cette analyse, nous explorerons plusieurs de ces méthodes, en soulignant comment chacune d'entre elles peut être adaptée en fonction des caractéristiques spécifiques d'un modèle ou des performances recherchées [8] [4].

4.3.1 BLEU

Aperçu :

Le BLEU Score, initialement conçu en 2002 par Papineni et al., émerge comme une métrique clé, initialement destinée à évaluer la qualité des traductions automatiques. Ce score, acronyme pour "Bilingual Evaluation Understudy," s'est progressivement étendu à d'autres domaines, dont la génération de code. Son objectif principal est de mesurer la similarité entre le code généré par un modèle et une solution de référence, en se basant sur la précision des mots et des phrases [5].

Processus d'Évaluation :

L'évaluation avec le BLEU Score suit un processus standard. Le modèle génère du texte en réponse à une tâche donnée. Des solutions de référence, généralement humaines, sont fournies pour la même tâche. Le BLEU Score mesure ensuite la similarité en comparant les n-grammes (séquences de n mots) du texte généré avec ceux des solutions de référence. Des scores sont ensuite attribués entre 0 et 1, où des scores plus élevés indiquent une meilleure similarité entre le texte généré et les références. La métrique tient compte de la précision des mots et des phrases, favorisant les sorties du modèle qui se rapprochent le plus des solutions de référence.

Défis:

Bien que le BLEU Score soit largement utilisé, il présente des défis importants : La métrique est sensible aux correspondances de n-grammes, ce qui peut négliger la correction sémantique du code généré. De plus, le BLEU Score ne tient pas compte de la qualité intrinsèque du code généré, se concentrant davantage sur la similarité lexicale. Il a aussi tendance à favoriser les séquences plus courtes, ce qui peut biaiser les résultats en faveur de générations de code succinctes. Aussi, en mettant l'accent sur la précision des n-grammes, le BLEU Score peut sous-estimer la complexité syntaxique et sémantique du code, limitant ainsi sa capacité à évaluer de manière exhaustive les compétences du modèle.

4.3.2 ROUGE

Aperçu :

Le ROUGE Score, acronyme pour "Recall-Oriented Understudy for Gisting Evaluation," a été développé pour évaluer la qualité des résumés automatiques de textes. Contrairement au BLEU Score, le ROUGE se concentre sur la rappel (recall), évaluant la capacité d'un modèle à rappeler l'information clé plutôt que de se focaliser sur la précision lexicale. Bien que son origine soit spécifique aux résumés, le ROUGE a également trouvé une utilité étendue dans diverses applications de traitement du langage naturel [5].

Processus d'Évaluation :

L'évaluation à l'aide du ROUGE suit un processus similaire à celui du BLEU. Le modèle génère un résumé automatique du texte source. Des résumés humains de référence sont fournis pour la comparaison. ROUGE évalue le rappel en comparant les n-grammes et les séquences de mots entre le résumé généré et les résumés de référence. ROUGE attribue des scores basés sur le rappel, mesurant à quel point le résumé généré couvre le contenu présent dans les résumés de référence. Il utilise diverses variantes telles que ROUGE-N pour les n-grammes et ROUGE-L pour la plus longue séquence commune.

$$\text{Rappel} = \frac{\text{Vrais Positifs}}{\text{Vrais Positifs} + \text{Faux Négatifs}} \quad (2)$$

Défis:

ROUGE peut être sensible aux variations linguistiques, ce qui peut conduire à des scores biaisés. En se concentrant sur le rappel, le ROUGE pourrait ne pas saisir entièrement la diversité lexicale et la qualité sémantique du texte généré. De plus l'évaluation du ROUGE dépend fortement de la qualité des résumés de référence humains.

4.3.3 HumaEval

Aperçu :

Le jeu de données HumanEval, élaboré par Chen et al. de chez OpenAI, constitue un élément crucial dans l'évaluation des modèles de LLM. Cette ressource cible spécifiquement l'évaluation de modèles entraînés pour la génération de code. L'évaluation de la correction des résultats est réalisée automatiquement grâce à des jeux de tests unitaires rédigés manuellement. Le jeu de données comprend un total de 164 problèmes de programmation originaux, chacun accompagné de docstrings et de tests unitaires associés [2].

Processus d'Évaluation :

La métrique utilisée pour évaluer les performances des modèles est le pass@k. Ce critère implique la génération de k échantillons de code par problème, et la considération d'un problème comme résolu dès lors qu'au moins l'un de ces échantillons satisfait les tests unitaires associés. La fraction totale de problèmes résolus est ensuite rapportée comme mesure de performance globale. Toutefois, il est important de noter que cette approche peut introduire une variance élevée dans les résultats. Afin de pallier ce défi, une méthode d'estimation non biaisée est mise en œuvre.

Concrètement, n échantillons de code (où n est supérieur ou égal à k, par exemple $n = 200$ et $k \leq 100$) sont générés pour chaque problème. Le nombre de réponses correctes c parmi ces échantillons est ensuite compté. Cette approche permet d'obtenir un estimateur non biaisé de la performance du modèle, offrant ainsi une évaluation plus fiable et robuste de sa capacité à résoudre les problèmes de programmation spécifiés dans le jeu de données [3].

$$\text{pass}@k := \bar{E}_{\text{Problèmes}} \left[1 - \frac{\frac{n-c}{k}}{\frac{n}{k}} \right] \quad (3)$$

Stratégies:

Il existe des stratégies spécifiques pour améliorer la génération de code lors de l'évaluation des modèles de LLM sur le jeu de données HumanEval.

L'une de ces astuces consiste à produire plusieurs échantillons de code par problème et à en choisir un pour l'évaluation. La heuristique recommandée pour cette sélection est de choisir l'échantillon ayant la plus haute moyenne de log-probabilité.

Un autre aspect crucial est l'ajustement du paramètre de "température" en fonction de k , où la température est un hyperparamètre qui module le degré de hasard dans la génération du code. Pour $k=1$, une température de 0.2 est recommandée, favorisant une sortie plus déterministe. En revanche, pour $k=100$, une température de 0.8 est préconisée pour encourager une plus grande diversité tout en maintenant une certaine cohérence comme vu dans la figure 9.

L'utilisation de l'échantillonnage du noyau avec un seuil $p=0.95$ est également suggérée pour affiner le processus de génération. Il est à noter que la conception de la requête (prompt) ne fait pas l'objet de stratagèmes particuliers dans le document. Ces astuces témoignent d'une approche nuancée visant à optimiser la qualité et la diversité des réponses générées, démontrant ainsi une compréhension approfondie du rôle de la température dans la régulation de la sortie des modèles de langage génératif.

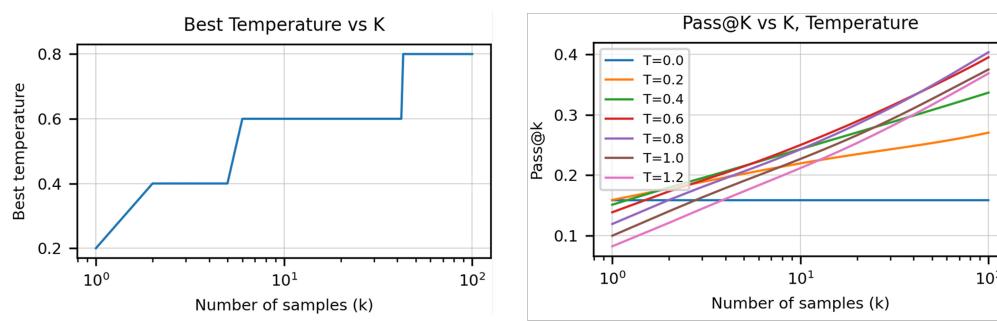


Figure 9: Relation entre la température et la métrique k .

Avantages et défis:

Le modèle d'évaluation présente des avantages significatifs. Tout d'abord, il mesure des compétences significatives en évaluant la capacité des modèles de langage génératif à générer des solutions de code pertinentes et fonctionnelles pour des problèmes de programmation spécifiques. De plus, l'évaluation est automatisée, ce qui permet une évaluation rapide et reproductible des performances des modèles.

En outre, ce modèle d'évaluation aborde certains des problèmes associés aux approches plus anciennes, notamment l'utilisation de métriques basées sur la correspondance, telles que BLEU avec des solutions de référence. Ces approches antérieures pouvaient présenter des limitations en termes de mesure de la qualité réelle des réponses générées.

Cependant, il convient de noter quelques inconvénients potentiels du modèle. Tout d'abord, la taille du jeu de données est relativement petite, ce qui le rend susceptible au surajustement des modèles. De plus, la portée du modèle est limitée, se concentrant uniquement sur des fonctions Python individuelles plutôt que sur des programmes complets. En outre, il existe un risque de contamination potentielle, notamment si les modèles sont entraînés sur des données provenant de nouvelles collectes sur GitHub, ce qui pourrait introduire des biais ou des changements dans la distribution des problèmes de programmation évalués.

4.3.4 Autres méthodes d'évaluation

Il existe bien sûr un nombre de méthodes d'évaluation similaires à HumanEval [3].

La base de données APPS est composée de 5000 exemples d'entraînement et 5000 exemples de test. Il se distingue par sa focalisation sur des problèmes de codage complexes, offrant ainsi une diversité de

défis pour évaluer les performances des modèles de langage génératif.

Le MBPP est composé de 974 tâches de programmation plus faciles à résoudre. Il représente une alternative permettant d'évaluer la capacité des modèles à résoudre des problèmes moins complexes, ce qui peut offrir un aperçu différent des compétences des modèles par rapport aux défis plus complexes.

Le MultiPL-E se démarque en traduisant les jeux de données HumanEval et MBPP dans 18 langages de programmation différents. Cette approche étend la portée de l'évaluation au-delà du seul langage Python, offrant ainsi une perspective sur la généralisation des compétences des modèles à travers différentes langues de programmation.

Enfin, le DS-1000 se base sur des questions posées sur Stack Overflow, comprenant 1000 problèmes de science des données liés à 7 bibliothèques Python telles que NumPy et Pandas. Cette approche évalue les performances des modèles sur des problèmes spécifiques liés à des domaines d'application concrets.

5 Shéma d'ensemble et finalité prévue :

5.1 Shéma d'ensemble :

Notre solution 'CodePoint AI' sera un outil crucial pour améliorer la productivité des développeurs, en réduisant le temps nécessaire pour comprendre et travailler avec de nouveaux codes ou architectures. En rendant l'intelligence artificielle accessible et personnalisable, nous ouvrons la voie à des développements logiciels plus efficaces, innovants et intuitifs. Le schéma ci-dessous donne un aperçu de la solution prévue par notre équipe :

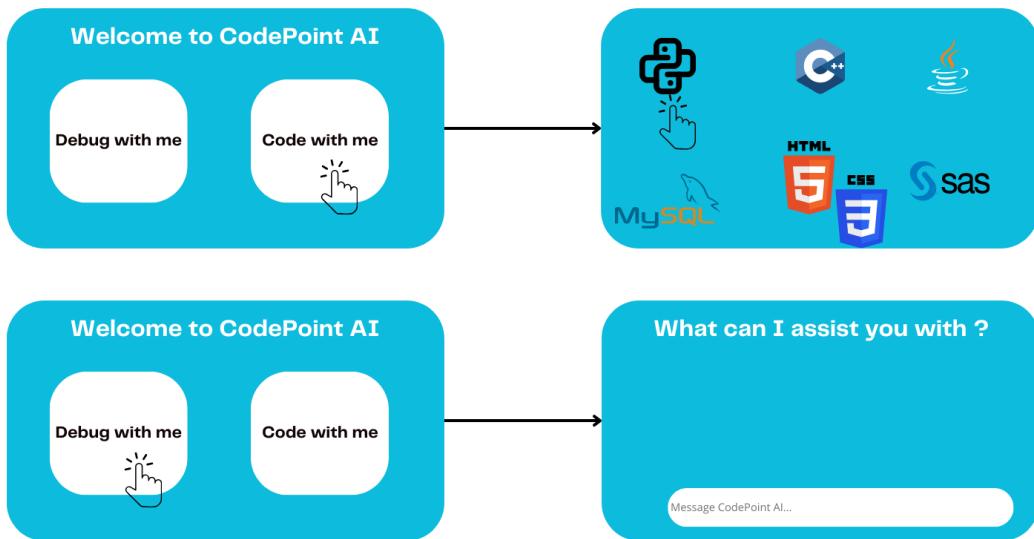


Figure 10: Schéma descriptif de la solution envisagée

Fonctionnalités Clés:

- Personnalisation Profonde: Les utilisateurs pourront personnaliser les modèles de langage pour qu'ils s'alignent mieux avec la syntaxe et les paradigmes spécifiques de leur choix de langage de programmation.
- Interface Utilisateur Intuitive: Inspirée de l'interface de GPT, notre interface sera conçue pour être familière aux développeurs, leur permettant de saisir des prompts et des requêtes de manière naturelle et directe.
- Potentiel Commercial: OnePoint CodeTuner AI sera développé avec une vision commerciale, en prévoyant la possibilité de le proposer à des utilisateurs externes et d'étendre ainsi notre impact sur le marché. Documentation Exhaustive: Le projet sera accompagné d'une documentation détaillée, expliquant non seulement comment utiliser l'interface, mais aussi comment les modèles ont été ajustés et comment ils peuvent être déployés efficacement.
- Prêt pour le Déploiement: Nous nous engageons à fournir des modèles bien documentés et prêts à l'intégration dans les pipelines de développement de logiciels existants, assurant ainsi une transition en douceur et une mise en œuvre rapide.

5.2 Diagramme de Gantt :

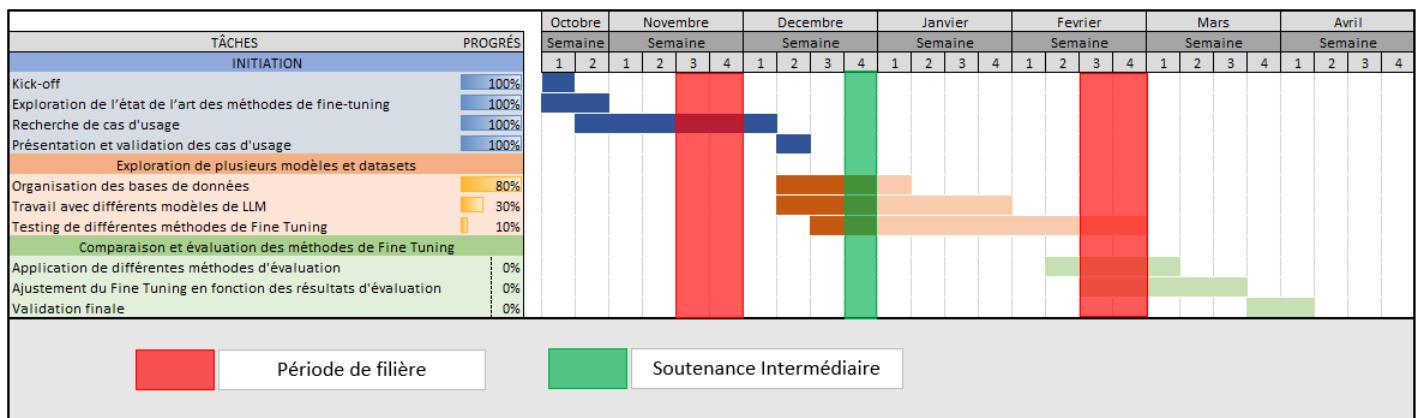


Figure 11: Gantt Chart du projet

6 Etat d'avancement et résultats obtenus :

6.1 Fine tuning sur les langages de programmation :

Après avoir fait les recherches et la documentation sur l'état de l'art et techniques de FineTuning on a décidé de mettre en pratique ces techniques .

Pour répondre aux critères du cahier des charges, le modèle Mistral 7B Instruct a été sélectionné pour sa légèreté et ses performances parmi les Lightweight Language Model (LLM) decoders. Pour adapter ce modèle à notre projet, nous avons employé des méthodes avancées de fine-tuning telles que Lora et Qlora avec la bibliothèque PEFT. Ces méthodes ont été appliquées à des ensembles de données financières et Python, visant à améliorer les capacités du LLM à générer du code.

Le processus de Fine tuning commence par un prétraitement et une préparation minutieuse des données d'entraînement pour s'adapter au format du modèle LLM utilisé. Ensuite, le modèle pré-entraîné a été chargé avec les bibliothèques PEFT, Lora et SFT Trainer pour l'entraînement.

Pendant l'entraînement, les fonctions de perte ont été affichées à chaque itération pour suivre la progression de l'apprentissage. Le modèle fine-tuned a été enregistré à la fin de cette phase. Les performances du modèle fine-tuned ont été évaluées par rapport au modèle de base lors d'essais d'inférence, permettant ainsi une comparaison qualitative directe de ses performances.

6.1.1 Fine tuning sur un dataset de Finance :

Dans ce premier travail on a choisi comme sujet de finetuning la Finance pour améliorer les réponses de notre modèle par rapport aux prompts sur ce sujet car les réponses du modèle de base étaient très peu satisfaisant et même aléatoires, comme on peut voir dans le **notebook** de ce travail .

On peut voir que l'output ne répond pas à la question posée et génère des réponses aléatoires mais proches du sujet. Nous avons donc trouvé un **dataset de Finance** composé de 69k lignes afin d'améliorer les performances de notre modèle par rapport ce sujet. Ce dataset comme son nom l'indique est créé pour entraîner le modèle Alpaca, Nous avons donc fait un travail de formatting et preprocessing afin de transformer le dataset dans le bon format pour entraîner notre modèle *Mistral7B-instruct* (voir le **notebook** dédié).

6.1.2 Fine tuning sur Python :

Dans ce premier travail on a choisi comme dataset d'entraînement **Python Code instructions**, qui est un dataset très utilisé pour la génération de code en Python composé de 18.6k lignes contenant des questions et réponses sous forme de code python ainsi que son input et output. Ce dataset est adapté au modèle Alpaca (voir le **notebook** dédié).

Nous avons tenté une inférence avec le modèle de base en lui demandant de générer des fonctions en Python, cependant, aucun résultat de code en Python n'a été généré. En revanche, après le processus de fine-tuning, le modèle a pu correctement produire le code demandé en utilisant les instructions spécifiées en prompt.

6.1.3 Fine tuning sur un nouveau langage de programmation :

Nous avons aussi tenté de fine tune les modèles identifiés sur un nouveau langage de programmation et observer les résultats de cette approche. Le langage de programmation sur lequel nous avons tenté cette approche est 'OPL' (Open Programming Language) qui est un langage de programmation pour les systèmes embarqués et les appareils mobiles qui exécutent les systèmes d'exploitation EPOC et Symbian. Il a été publié par la société britannique Psion en 1984.

Le dataset (Lien vers le **dataset**) était déjà bien pré-traité et convenait parfaitement aux normes de *Mistral-7B-Instruct* et *Llama-2-7b-chat-hf* (**Normes d'instructions pour les modèles**) grâce à M. Chris Hay sur Hugging Face.

Sans surprise, dans un premier temps, lorsque l'on demande aux modèles de générer un simple code OPL, les modèles répondent ne pas connaître ce langage de programmation et sont donc dans l'incapacité

de générer un code 'Hello World'. Après le processus de fine-tuning, les modèles génèrent un code qui respecte les normes d'OPL et répondent correctement aux questions sur la documentation d'OPL ainsi qu'aux informations sur ses origines.

Pour cette approche nous n'avons pu faire pour l'instant qu'une mesure qualitative des résultats obtenus en comparant l'output avant et après fine tuning, nous envisageons de trouver les métriques idéales pour juger quantitativement la qualité de fine tuning.

Les résultats sont disponibles dans la branche **@enaouram** de notre Repo GitHub.

7 Conclusion et tâches futures :

7.1 Conclusion générale

Les recherches approfondies, la documentation minutieuse et la mise en pratique du fine-tuning sur des modèles de langage ont généré des améliorations significatives dans nos compétences de fine-tuning. Ces progrès se sont concrétisés par des résultats remarquables dans la génération de code Python, Finance, et même dans des langages de programmation moins connus comme OPL. L'utilisation de méthodes avancées telles que LoRA, Qlora et PEFT a joué un rôle essentiel en affinant les performances de nos modèles pour des tâches spécifiques. Ces résultats attestent de l'efficacité de nos techniques de fine-tuning sur des jeux de données plus simples et légers, démontrant ainsi notre capacité à atteindre les objectifs fixés avec notre tuteur de projet.

7.2 Problèmes rencontrés :

Au cours du développement de notre projet de fine-tuning pour les modèles de langage dédiés à la programmation, notre équipe a rencontré plusieurs défis qui ont influencé notre progression et notre stratégie de déploiement. Voici les principales difficultés auxquelles nous avons dû faire face :

- Définition Initiale du Projet: Le manque de clarté autour du sujet au début du projet a conduit à une phase de redéfinition prolongée. Nous avons consacré un temps considérable à affiner le problème et à identifier les cas d'usage les plus pertinents, ce qui a retardé les phases initiales de conception et de planification.
- Collecte de Données: La recherche et l'acquisition de jeux de données appropriés se sont révélées être des tâches extrêmement chronophages. La disponibilité et la qualité des datasets nécessaires pour l'entraînement et le fine-tuning ont posé des défis significatifs.
- Stabilité Environnementale: Nous avons été confrontés à des problèmes d'instabilité avec l'environnement Mydock, notamment des erreurs relatives à la non-détection de GPU. Nous tenons à remercier M. Paul Bizouard pour sa réactivité et son soutien dans la mise à jour des machines virtuelles que nous utilisons.
- Contraintes de Ressources: Les tests de performance ont été entravés par des limitations en ressources matérielles, notamment avec Google Colab qui offre une Tesla T4 avec 16 Go de VRAM. Même avec l'accès récent à une nouvelle machine virtuelle équipée d'une Tesla V100 avec 32 Go de VRAM, nous avons été confrontés à des contraintes de ressources insuffisantes.

- Définition des Métriques: Un défi notable a été l'établissement de métriques adaptées pour évaluer la qualité des codes générés par les modèles pour des langages de programmation autres que Python. La mesure de la performance des modèles de l'auto-complétion et de la génération de code nécessite une approche méthodologique et des outils de mesure qui doivent être affinés pour chaque langage spécifique.

7.3 Tâches futures :

Maintenant que nous avons obtenu des résultats prometteurs, il est temps d'appliquer ces méthodes à des cas d'utilisation plus complexes, conformément aux objectifs convenus lors des réunions de suivi de projet :

- **Fine-tuning avec Stackoverflow** : Utiliser le dataset de Stackoverflow pour entraîner nos modèles, en se concentrant sur la résolution d'erreurs et de problèmes spécifiques rencontrés par les développeurs chez Onepoint. Cela aiderait à répondre à des tâches de déboggage et à fournir un assistant précis et rapide aux développeurs.
- **Fine-tuning pour SAS** : Nous avons déjà trouvé un dataset "**THE STACK**" qui contient des codes sources en plusieurs langages de programmation (300 langages en total) y compris le langage SAS. Nous avons fait l'exploration de ce dataset pour confirmer sa conformité pour notre Use Case. Il reste donc à préprocesser le dataset pour extraire les données relatives au langage SAS , car le dataset est très large, puis réaliser le fine-tuning sur ce dataset. Cela permettra de répondre aux besoins spécifiques en matière de génération de code dans le language SAS chez Onepoint et pour lequel il n'existe pas de solution actuellement.
- **Exploration de nouvelles méthodes d'évaluation** : Rechercher et tester de nouvelles méthodes pour évaluer la qualité des modèles fine-tuned. Cela pourrait impliquer l'exploration de métriques plus précises ou spécifiques à des contextes linguistiques en général et à la génération de code en particulier pour évaluer les performances de nos modèles.

8 Références :

References

- [1] “Large Language Model Evaluation in 2023: 5 Methods,” research.aimultiple.com. <https://research.aimultiple.com/large-language-model-evaluation/>
- [2] “HumanEval: LLM Benchmark for Code Generation,” Deepgram. <https://deepgram.com/learn/humaneval-llm-benchmark> (accessed Dec. 18, 2023).
- [3] X. Du et al., “ClassEval: A Manually-Crafted Benchmark for Evaluating LLMs on Class-level Code Generation.” Available: <https://arxiv.org/pdf/2308.01861.pdf>
- [4] “Papers with Code - Evaluating Large Language Models Trained on Code,” paperswithcode.com. <https://paperswithcode.com/paper/evaluating-large-language-models-trained-on> (accessed Dec. 18, 2023).
- [5] A. sahaymaniceet, “Large Language Model (LLM) Evaluation Metrics – BLEU and ROUGE,” Jul. 08, 2023. <https://mlexplained.blog/2023/07/08/large-language-model-llm-evaluation-metrics-bleu-and-rouge/> (accessed Dec. 18, 2023).
- [6] R. Li et al., “StarCoder: may the source be with you!,” arXiv (Cornell University), May 2023, doi: <https://doi.org/10.48550/arxiv.2305.06161> (accessed Dec. 7, 2023).
- [7] V. Lalin, V. Deshpande, and A. Rumshisky, “Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning,” arXiv.org, Mar. 27, 2023. <https://arxiv.org/abs/2303.15647> (accessed Dec. 18, 2023).
- [8] E. J. Hu et al., “LoRA: Low-Rank Adaptation of Large Language Models,” arXiv:2106.09685 [cs], Oct. 2021. Available: <https://arxiv.org/abs/2106.09685> (accessed Dec. 18, 2023).