Université de Montpellier
Faculté des science
Département Informatique

Master 1 : Intelligence artificielle et science de données

HAI823I -TER DE MASTER 1

Few-shot learning with Large Language Models (LLMs) for Natural Language Inference (NLI) tasks for French

---

Réalisé par :

- Amira Djihane Merad
- Karima Bouaboud

Encadré par :

- Maximos Skandalis
- Richard Moot
- Michael Sioutis

Année universitaire : 2024/2025

Table des matières

Table des figures

Abréviations

TALN	Traitement Automatique du Langage Naturel
NLI	Natural Language Inference
LLM	Large language model
MAML	Model-Agnostic Meta-Learning
MLM	Masked Language Models
PEFT	Parameter-Efficient Fine-Tuning
PLM	Pre-trained language models
EFL	Entailment as Few-shot Learner
LoRa	Low Rank Adapter

Introduction

Dans le domaine du traitement automatique du langage naturel (TALN), l'apprentissage en *few-shot*, ou apprentissage à partir de peu d'exemples, suscite un intérêt croissant. Cette approche se révèle particulièrement précieuse dans des contextes où les données annotées sont rares ou coûteuses à produire. Grâce à l'émergence de modèles de langage préentraînés tels que BERT, RoBERTa ou GPT-3, il devient possible d'adapter des modèles puissants à de nouvelles tâches, en ne s'appuyant que sur quelques exemples annotés par classe.

Parmi les tâches emblématiques du TALN où le *few-shot* learning démontre un fort potentiel, la classification de paires de phrases et plus spécifiquement le cadre du Natural Language Inference (NLI) occupe une place centrale. Dans ce cadre, l'objectif est de déterminer la relation logique qui unit une prémisse à une hypothèse, selon trois étiquettes principales : *entailment* (implication), contradiction, ou neutralité. Comprendre si deux phrases se valident, se contredisent, ou sont sans lien logique explicite constitue une compétence fondamentale, applicable à une large gamme de cas concrets tels que la détection de duplicatas, la vérification de faits, ou encore le raisonnement automatique.

C'est dans ce contexte que s'inscrit notre projet, dont l'objectif a été de construire un modèle *few-shot* performant pour la tâche de *NLI*, en tirant parti de la puissance des modèles de langage préentraînés. Nous avons exploré différentes méthodes récentes, notamment SetFit, qui allie efficacité et simplicité d'implémentation en se fondant sur la génération d'exemples par similarité sémantique et l'entraînement contrastif.

Afin d'évaluer la robustesse et la généralisation de notre approche, nous avons testé notre modèle sur plusieurs jeux de données de NLI en français et en anglais, aux caractéristiques variées :

- RTE3, un corpus standard pour l'inférence textuelle,
- GQNLI, une version française adaptée de QNLI annotée manuellement,
- SICK-FR-mt, une traduction automatique du corpus SICK portant sur les inférences logiques,
- et FRACAS, un jeu de données issu de la logique formelle contenant des inférences complexes.
- DACCORD , un jeu de données axé sur les phénomènes linguistiques (quantificateurs, négations,etc).

À travers ce travail, nous avons cherché à évaluer dans quelle mesure l'apprentissage en few-shot permet de tirer le meilleur parti de ressources limitées, tout en garantissant des performances satisfaisantes sur une tâche aussi fine et exigeante que celle de l'inférence textuelle.

Première partie

État de l'art

Chapitre 1

Concepts clés et fondements théorique

1.1 Traitement automatique du langage naturel (TALN)

Dans cette section, nous introduisons les bases du Traitement Automatique du Langage Naturel (TALN), un domaine à la croisée de la linguistique, de l'informatique et de l'intelligence artificielle. Le TALN a pour objectif de permettre aux machines de comprendre, d'analyser et de produire du langage humain, aussi bien à l'écrit qu'à l'oral.

L'idée est de créer des systèmes capables d'interagir avec les humains en utilisant notre langue de manière naturelle et pertinente. cependant il existe une différence entre le TALN et la linguistique informatique ,cette dernière utilise des outils numériques pour étudier les langues, tandis que le TALN cherche surtout à améliorer la communication entre l'homme et la machine en s'appuyant sur des modèles et des algorithmes adaptés au langage naturel. Il existe deux grandes approches pour traiter le langage naturel :

L'approche symbolique :
qui repose sur des règles précises, des grammaires formelles, et des connaissances linguistiques explicites. C'est une approche plus "manuelle", souvent complexe à mettre en œuvre.

L'approche statistique :
plus moderne, qui s'appuie sur de grandes quantités de données textuelles pour entraîner des modèles à reconnaître des patterns. Grâce à l'essor de l'apprentissage automatique et surtout de l'apprentissage profond (*deep learning*), cette méthode est aujourd'hui la plus utilisée. Des modèles comme BERT ou GPT en sont de très bons exemples.

La reconnaissance d'implication textuelle est une tâche fondamentale en traitement automatique du langage naturel qui vise à déterminer si une relation logique existe entre deux segments de texte : une prémisse et une hypothèse. Plus précisément, il s'agit de vérifier si la prémisse implique, contredit ou est neutre par rapport à l'hypothèse. Cette tâche est essentielle pour comprendre le sens profond des textes et a des applications très variées dans des domaines comme la compréhension de documents, les systèmes de questions-réponses, le résumé automatique, ou encore la détection de fausses informations.

Ce dernier permet d'automatiser l'évaluation de la cohérence entre différentes phrases ou documents, facilitant ainsi des systèmes capables de raisonner sur le langage naturel de manière plus fine et contextuelle.

1.1.1 Inférence textuelle (*Natural Language Inference*)

L'inférence textuelle, qu'on appelle souvent NLI (*Natural Language Inference*), c'est une tâche en traitement automatique du langage qui sert à vérifier s'il y a une relation logique entre deux phrases : une prémisse et une hypothèse. Le principe est de voir comment ces deux phrases se répondent ou se contredisent. Il y a trois cas possibles :

- Implication (*Entailment*) : si la prémisse est vraie, alors l'hypothèse l'est forcément aussi.
- Contradiction : les deux phrases se contredisent , elles ne peuvent pas être vraies en même temps.
-
- Cas neutre : Il ne peut pas être conclu que l'hypothèse est vraie ou fausse à partir des informations de la prémisse.

Cette tâche est importante en TALN , parce qu'elle ne se contente pas de comprendre les mots : elle pousse les modèles à raisonner, à faire le lien entre les idées et à prendre en compte le contexte pour vraiment "comprendre" ce qui est dit. Cette tâche est devenue un *benchmark* central en TALN, car elle exige du modèle non seulement une compréhension lexicale, mais aussi une capacité à raisonner sur le contenu, les relations sémantiques, et le contexte [2].

1.2 Les grands modèles de langage (*LLMs*)

Les grands modèles de langue, ou LLMs (*Large Language Models*), sont des modèles puissants capables de comprendre et de générer du texte de manière fluide et naturelle. Reposant sur l'architecture des *Transformers* [1], ils ont profondément transformé le domaine du traitement automatique du langage naturel. Ces modèles s'appliquent à une grande variété de tâches linguistiques sans entraînement spécifique.

1.2.1 Apprentissage et Entraînement

Le développement des LLMs repose sur plusieurs approches d'apprentissage complémentaires :

- **Apprentissage auto-supervisé** : pré-entraînement massif sur des corpus ouverts (ex. : Wikipédia, Common Crawl), basé sur la prédiction du mot suivant.
- **Apprentissage supervisé** : ajustement du modèle à des tâches précises (classification, résumé, etc.).
- **Apprentissage par renforcement avec retour humain (RLHF)** : alignement du comportement du modèle avec les attentes humaines.

1.2.2 Architecture des Transformers

Les *Transformers* Les Transformers sont au cœur des modèles de langage de grande taille (LLMs). Leur coquille permet de traiter efficacement de longues séquences textuelles grâce à une structure modulaire. Celle-ci peut inclure un encodeur, un décodeur, ou les deux, selon les modèles. Par exemple, BERT repose uniquement sur une architecture d'encodeur, tandis que GPT utilise uniquement un décodeur. Tous les modèles basés sur

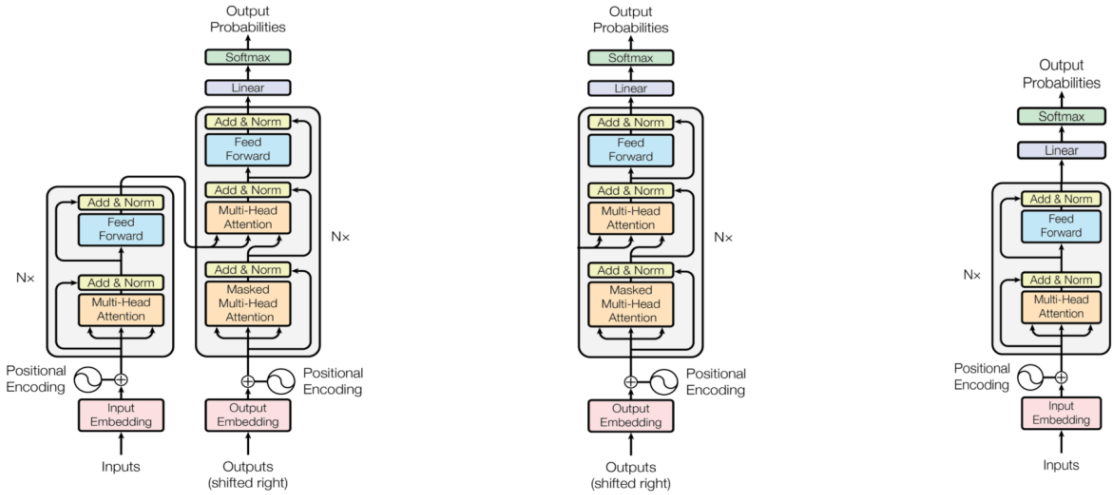

Figure 1.1 – architecture des *Transformers*[1]

les *Transformers* ne combinent donc pas nécessairement les deux composants. La figure 1.1 illustre plus en détail la structure de cette architecture.

Représentation du texte

Avant d'être traité, le texte est converti en vecteurs numériques (*embeddings*). Un encodage positionnel est ajouté pour informer le modèle de l'ordre des mots.

L'encodeur

Chaque encodeur est composé de plusieurs couches identiques, contenant :

- **Attention multi-tête** : ce mécanisme permet au modèle de se concentrer simultanément sur différentes parties d'une séquence, en capturant diverses relations contextuelles entre les mots. Chaque tête d'attention effectue un calcul de type *scaled dot-product attention*, défini par la formule suivante :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

  où :
  - Q (*queries*), K *(keys)* et V (*values*) sont des matrices obtenues par projection linéaire des entrées,
  - d_k est la dimension des vecteurs clés (keys),
  - le produit scalaire $QK^\top$ mesure la similarité entre les éléments de la séquence,
  - la division par $\sqrt{d_k}$ permet de stabiliser les gradients,
  - la fonction *softmax* génère des poids d'attention normalisés.

  Plusieurs têtes d'attention sont utilisées en parallèle pour apprendre différentes relations, puis leurs sorties sont concaténées et projetées à nouveau pour produire une représentation combinée.
- **Réseau à propagation avant (*feed-forward*)** : appliqué de manière indépendante à chaque position de la séquence, ce sous-réseau est composé de deux

couches linéaires séparées par une fonction d'activation non linéaire, généralement ReLU ou GELU. Il permet de transformer et d'enrichir les représentations issues de l'attention, en introduisant une capacité de modélisation plus complexe à chaque position.

La transformation effectuée par ce sous-réseau peut être exprimée par la formule suivante :

$$\mathrm{FFN}(x) = f(xW_1 + b_1)W_2 + b_2$$

où :

- x est le vecteur d'entrée correspondant à une position donnée dans la séquence,
- $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ et $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ sont les matrices de poids,
- b_1 et b_2 sont les biais associés,
- f est une fonction d'activation non linéaire, typiquement ReLU ($\max(0, \cdot)$) ou GELU,
- d_{model} est la dimension du modèle, et d_{ff} est la dimension intermédiaire (souvent plus grande, ex : $4\times$ d_{model}).

— ***Add & Norm*** : ce mécanisme ajoute les connexions résiduelles (skip connections) entre l'entrée et la sortie de chaque sous-couche (attention ou feed-forward), suivies d'une normalisation couche par couche (*Layer Normalization*). Cela stabilise et accélère l'apprentissage, en facilitant la propagation du gradient dans les réseaux profonds.

— ***Add & Norm***

Cette couche combine une connexion résiduelle et une normalisation de couche pour stabiliser et faciliter l'apprentissage.

Soit $\mathbf{x}$ l'entrée et $\mathcal{F}(\mathbf{x})$ la transformation appliquée (ex. couche d'attention ou réseau *feed-forward*).

La sortie est donnée par :

$$\mathbf{y} = \mathrm{LayerNorm}(\mathbf{x} + \mathcal{F}(\mathbf{x}))$$

où :

- $\mathbf{x} + \mathcal{F}(\mathbf{x})$ correspond à la connexion résiduelle, qui ajoute l'entrée à la transformation pour conserver l'information initiale.
- La normalisation de couche standardise ce vecteur via :

$$\mathrm{LayerNorm}(\mathbf{z}) = \frac{\mathbf{z} - \mu}{\sigma} \odot \gamma + \beta$$

avec

$$\mu = \frac{1}{H}\sum_{i=1}^{H} z_i, \quad \sigma = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(z_i - \mu)^2 + \epsilon}$$

- μ est la moyenne, σ l'écart-type des composantes de $\mathbf{z} = \mathbf{x} + \mathcal{F}(\mathbf{x})$.
- γ et β sont des paramètres appris permettant de réajuster l'échelle et la position après normalisation.

Appliquer afin d'améliorer la stabilité et la vitesse d'entraînement en évitant la dégradation des gradients et en uniformisant les activations.

Le décodeur

Le décodeur ajoute deux mécanismes supplémentaires :

— **Attention masquée** : utilisée dans les décodeurs de type GPT, cette variante du mécanisme d'attention empêche chaque position de la séquence de « voir » les mots futurs. Elle garantit ainsi une génération strictement séquentielle, mot par mot. Cela est crucial pour la génération de texte autoregressive.

Techniquement, un masque (matrice triangulaire) est appliqué avant la fonction softmax afin de forcer les positions futures à être ignorées lors du calcul des poids d'attention :

$$\text{MaskedAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right)V$$

où :

— Q, K, V sont les matrices de requêtes, clés et valeurs,

— d_k est la dimension des clés,

— M est une matrice de masque contenant $-\infty$ pour les positions interdites (positions futures), et 0 ailleurs.

Le terme $+M$ dans le *softmax* agit comme une barrière : les positions masquées deviennent nulles après l'application de la fonction *softmax*.

— **Attention croisée** : Cette forme d'attention permet au décodeur de s'appuyer sur les représentations produites par l'encodeur. Elle relie chaque position du décodeur à toutes les positions de la séquence source, facilitant ainsi le transfert d'information entre les deux modules.

Contrairement à l'attention auto-régressive où requêtes, clés et valeurs proviennent de la même séquence, ici les **requêtes** (Q) proviennent du décodeur, tandis que les **clés** (K) et **valeurs** (V) proviennent de l'encodeur :

$$\text{CrossAttention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

où :

— Q : projections des représentations du décodeur,

— K, V : projections des représentations de sortie de l'encodeur,

— d_k : dimension des clés.

Cela permet au décodeur de concentrer son attention sur les parties pertinentes de la séquence d'entrée pour générer chaque mot de sortie.

Génération finale

La sortie du dernier bloc du décodeur est d'abord projetée dans l'espace du vocabulaire via une couche linéaire. Ensuite, une fonction *softmax* est appliquée pour transformer ces scores (logits) en une distribution de probabilité sur l'ensemble des mots du vocabulaire. Le mot le plus probable est ensuite sélectionné (par exemple via l'*argmax*) et ajouté à la séquence générée.

La fonction softmax est définie comme suit :

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{|V|} e^{z_j}}$$

où :
— z_i est le logit associé au mot i,
— $|V|$ est la taille du vocabulaire,
— le dénominateur assure que la somme des probabilités vaut 1.

Ce processus est répété de manière *autoregressive* : à chaque étape, le mot généré est réinjecté comme entrée pour produire le suivant, jusqu'à atteindre un token spécial de fin ou une longueur maximale.

1.3 Points forts des LLMs

Les grands modèles de langage sont particulièrement puissants grâce à :
— leur **taille colossale**, avec des centaines de milliards de paramètres,
— un **entraînement sur des corpus multilingues et multidomaines**,
— la capacité à effectuer des tâches sans apprentissage supplémentaire (*zero-shot* et *few-shot learning*).

1.3.1 L'apprentissage avec peu d'exemple *(few-shot)*

Le *few-shot learning* est une approche qui se distingue de l'apprentissage supervisé classique, car elle ne nécessite que très peu d'exemples annotés pour une nouvelle tâche. L'idée repose sur le fait que les modèles de langage pré-entraînés comme GPT, BERT ou T5 ont déjà appris une grande quantité de connaissances sur des corpus très variés, et peuvent donc s'adapter rapidement à de nouvelles tâches avec un minimum d'exemples.

Les différentes formes de *few-shot*

On peut mettre en œuvre le *few-shot learning* de différentes manières, selon le contexte et les ressources disponibles :
— ***Few-shot* par *prompting* (ou apprentissage dans le contexte)** : cette méthode consiste à formuler des exemples directement dans l'instruction envoyée au modèle. Par exemple, on montre quelques paires question-réponse, puis on demande au modèle de compléter ou de répondre à une nouvelle question dans le même style.
— ***Few-shot* avec un léger *fine-tuning*** : ici, le modèle est ajusté avec un petit jeu de données spécifique à la tâche. Cela demande un peu plus de ressources que le prompting, mais peut donner de meilleurs résultats pour des cas complexes ou sensibles.
— ***Few-shot* par méta-apprentissage** : le modèle est entraîné non pas pour résoudre une tâche directement, mais pour apprendre à s'adapter rapidement à n'importe quelle tâche. Une méthode connue dans ce domaine est MAML (*Model-Agnostic Meta-Learning*). L'objectif est de rendre le modèle capable d'apprendre efficacement, même avec très peu de données.

1.4 Implication Textuelle avec l'apprentissage avec peu d'exemples

Dans cette section, nous allons introduire le travail de [3] :

1.4.1 Contexte

Les récents progrès en traitement automatique du langage naturel (TALN) reposent sur le paradigme pré-entraînement *fine-tuning*, où un modèle est d'abord pré-entraîné sur de grands corpus textuels puis adapté à des tâches spécifiques à l'aide de données annotées. Toutefois, ce paradigme dépend fortement de la disponibilité de jeux de données labellisés, ce qui pose problème en pratique.

Le *few-shot learning* répond à cette contrainte en apprenant à partir de très peu d'exemples. Des approches telles que le *prompt-based learning* avec GPT-3 se sont révélées prometteuses, mais elles restent coûteuses en calcul. D'autres méthodes proposent de reformuler les tâches en complétion (*Masked Language Modeling*) , mais leur efficacité baisse face à des distributions divergentes entre l'entraînement et le test.

1.4.2 Adaptation en tâche d'implication

Une approche qui consiste à reformuler les tâches NLP comme des problèmes d'inférence textuelle (*textual entailment*). Chaque classe est représentée par une phrase descriptive, et l'objectif est de déterminer si une phrase d'entrée implique cette description.

1.4.3 Méthodologie *Entailment as Few-shot Learner (EFL)*

L'approche *Entailment as Few-shot Learner (EFL)* suit les étapes suivantes :

1. **Reformulation des classes** : chaque label est associé à une phrase descriptive
2. **Génération de paires** : pour chaque entrée , on crée une paire avec chaque
3. ***Fine-tuning*** : un modèle NLI est ajusté pour prédire l'implication.
4. **Prédiction** : la classe prédite est celle avec la plus forte probabilité d'implication.

1.4.4 Résultats expérimentaux

les résultats démontrent que la méthode *Entailment as Few-shot Learner* (EFL) surpasse les approches classiques basées sur le *masked language modeling* (MLM) et le simple *prompting* dans des contextes de *few-shot learning*. Cette supériorité s'explique par la reformulation des tâches de classification en problèmes d'inférence textuelle, ce qui permet au modèle de mieux exploiter ses capacités de raisonnement linguistique et sémantique. Par rapport au *fine-tuning* traditionnel, qui nécessite un ajustement important des paramètres sur des jeux de données relativement volumineux, EFL offre une meilleure généralisation avec un nombre très réduit d'exemples annotés. De plus, cette approche conserve une compétitivité notable même lorsqu'elle est appliquée en mode *full training*, tout en étant compatible avec des modèles plus compacts et *multilingual*. Ces caractéristiques font

de l'EFL une solution particulièrement flexible et efficace, capable de s'adapter à différents scénarios d'apprentissage où les ressources annotées sont limitées, ce qui est souvent le cas dans les applications réelles du traitement automatique du langage naturel.

Chapitre 2

Méthode de Réglage fin *Fine-Tuning*

Le réglage fin *fine-tuning* est une technique essentielle en apprentissage profond, qui consiste à adapter un modèle pré-entraîné à une tâche spécifique.[4] Plutôt que de former un modèle depuis zéro, le *fine-tuning* exploite les connaissances acquises lors d'un pré-entraînement sur de grandes quantités de données générales, et affine ces connaissances sur un jeu de données plus restreint et ciblé. Cette méthode permet d'obtenir des performances élevées tout en réduisant significativement les coûts en temps et en ressources. Plusieurs approches de *fine-tuning* existent, allant du réglage complet des paramètres du modèle à des méthodes plus économes en calcul, comme le *fine-tuning* efficace des paramètres (PEFT) ou l'adaptation des *Sentence Transformers* via SetFit.

2.1 La Méthode *SetFit* et le *fine-tuning* des Sentence Transformers

SetFit *(Sentence Embedding Fine-Tuning)* s'appuie sur les *Sentence Transformers* [5], des modèles pré-entraînés basés sur l'architecture *Transformer*, adaptés via des architectures de type *Siamese* [6] et *triplet networks*[7] . Ces architectures permettent d'apprendre des représentations de phrases (*embeddings*) qui reflètent la similarité sémantique : des phrases proches sémantiquement ont des *embeddings* proches, et inversement.

Les *Sentence Transformers* produisent un vecteur dense de dimension fixe représentant une phrase, utilisable par la suite par des algorithmes classiques de machine learning.

Approche *SetFit* pour la classification en *Few-Shot*

SetFit adopte une démarche en deux étapes :

Fine-tuning des *Sentences Transformer*

Pour compenser le faible nombre d'exemples labellisés (*few-shot*), SetFit utilise un apprentissage contrastif :

À partir d'un petit jeu de données comptant K exemples, on génère des triplets positifs où x_i et x_j sont deux phrases appartenant à la même classe c, ainsi que des triplets négatifs

$$(x_i, x_j, (0|1))$$

où x_i est dans la classe c et x_j dans une autre classe.

Cela permet de constituer un ensemble de paires contrastives T de taille R un hyperparamètre fixé par défaut à 20, et $|C|$ le nombre de classes.

Cette méthode augmente artificiellement la taille du jeu d'entraînement, passant de K exemples à environ

$$\frac{K(K-1)}{2}$$

paires, ce qui améliore la robustesse du *fine-tuning* en contexte *few-shot*.

Entraînement du classificateur

Après avoir affiné le *Sentence Transformer*, chaque phrase x_i est encodée en un *embedding*

$$\mathrm{Emb}_{x_i} = ST(x_i).$$

Ces *embeddings*, associés à leurs étiquettes y_i, constituent le jeu d'entraînement du classificateur final, généralement une régression logistique.

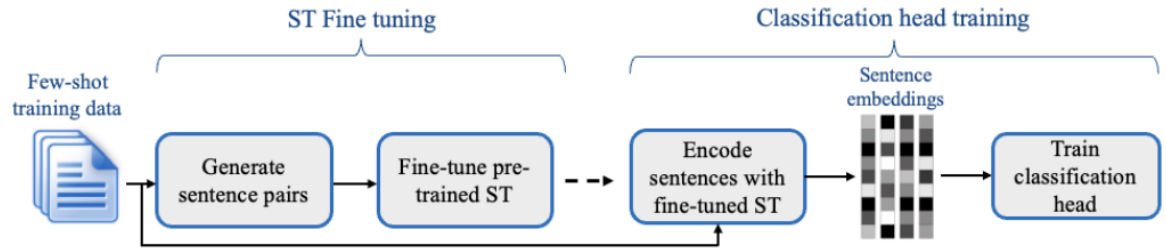

Figure 2.1 – Bloc d'entraînement du*fine-tuning* SetFit

Inférence

Lors de l'inférence, une phrase non vue x_i est encodée avec le Sentence Transformer finement ajusté.

Le classificateur prédit alors la classe correspondante à partir de l'*embedding* :

$$\hat{y}_i = CH(ST(x_i)),$$

où CH désigne la fonction de prédiction du classificateur. L'approche SetFit propose une manière simple et efficace d'entraîner des modèles de classification dans des contextes où les données annotées sont rares. Elle s'appuie sur un *fine-tuning* contrastif du modèle pré-entraîné `all-MiniLM-L6-v2`, combiné à un classificateur léger, comme la régression logistique. Cette stratégie permet non seulement d'augmenter artificiellement la quantité d'exemples via des paires contrastives, mais aussi d'affiner les représentations sémantiques des phrases, rendant le modèle plus robuste et performant.

Avec 8 exemples par classe, SetFit parvient à produire des résultats motivant. Par exemple, il atteint 87,5% de précision sur la base de données SST-2 (analyse de sentiments), 89,3% sur AG News, 95,2% sur TREC-6 (classification de questions), et jusqu'à 99,2% sur DBPedia. Ces performances sont d'autant plus impressionnantes qu'elles sont obtenues avec un coût de calcul bien moindre que les approches classiques de *fine-tuning* de modèles de type BERT.

2.2 Réglage Fin Efficace des Paramètres (PEFT)

Le réglage fin efficace des paramètres (PEFT, *Parameter-Efficient Fine-Tuning*) [8] est une méthode d'adaptation des grands modèles de langage (LLM) ou des réseaux neuronaux pré-entraînés à des tâches spécifiques, sans nécessiter un réentraînement complet. Cette technique vise à améliorer les performances tout en réduisant considérablement les coûts en temps, en calcul et en stockage.

Le PEFT consiste à geler la majorité des paramètres d'un modèle pré-entraîné et à n'entraîner qu'un petit sous-ensemble de paramètres dits adaptateurs, insérés généralement dans les couches supérieures. Cela permet de préserver les connaissances acquises lors du pré-entraînement tout en spécialisant le modèle pour la tâche cible.

Certaines méthodes PEFT utilisent également des techniques de contrôle des gradients, afin de limiter la mémoire utilisée pendant l'entraînement.

2.2.1 Principes du PEFT

Gel des paramètres pré-entraînés

Dans le cadre du PEFT, les poids originaux du modèle sont **figés**, c'est-à-dire qu'ils ne sont pas modifiés lors de l'entraînement. Seuls les paramètres ajoutés (adaptateurs, prompts, matrices LoRA, etc.) sont mis à jour. Ceci permet de :

— réduire drastiquement la dimension de l'espace de recherche,
— diminuer les besoins en mémoire et en calcul,
— conserver la stabilité des connaissances acquises lors du pré-entraînement.

Injection de modules spécifiques

Selon la méthode employée, des modules d'adaptation sont insérés dans le modèle :

Adaptateurs Petits modules (souvent deux couches linéaires séparées par une activation ReLU) ajoutés entre les blocs Transformer, qui apprennent à modifier les représentations internes sans toucher aux poids principaux.

Low Rank Adapter (LoRA) Traditionnellement, le *fine-tuning* d'un grand modèle de langage (comme BERT ou GPT-3) nécessite la mise à jour de tous les paramètres, ce qui est coûteux en ressources (mémoire, stockage, temps). Cette approche devient impraticable pour des modèles très volumineux, comme GPT-3 avec ses 175 milliards de paramètres [9].

Pour contourner cette limitation, plusieurs méthodes ont été développées, telles que les *adapters* ou le *prefix-tuning*. Une autre solution est la méthode LoRA (*Low-Rank Adaptation*).

Principe de LoRA Proposée par Hu et al. (2021) [9], LoRA vise à **réduire significativement le nombre de paramètres entraînés** et la mémoire requise, en **décomposant la mise à jour des poids en matrices de faible rang**, tout en maintenant les poids d'origine du modèle inchangés.

Les matrices de poids W des réseaux profonds, apprises lors du pré-entraînement, sont conservées fixes. LoRA introduit une mise à jour ΔW définie par :

$$\Delta W = A \cdot B,$$

où $A \in \mathbb{R}^{m \times r}$ et $B \in \mathbb{R}^{r \times n}$ sont des matrices de rang faible ($r \ll m, n$). Cette factorisation repose sur le théorème de factorisation en algèbre linéaire, qui affirme qu'une matrice de rang r peut s'exprimer comme le produit de deux matrices de dimensions respectives $m \times r$ et $r \times n$.

Ce principe s'appuie aussi sur la notion de **dimensionnalité intrinsèque** des modèles profonds, démontrée par une étude de Facebook AI Research (2020), selon laquelle l'espace utile de représentation est souvent de bien plus faible dimension que l'espace paramétrique total.

QLoRA

Variante optimisée de LoRA pour réduire la consommation mémoire :
— les poids gelés sont quantifiés en 4 bits,
— des matrices LoRA de faible rang sont ajoutées,
— le modèle peut être entraîné efficacement sur des machines avec des ressources modestes.

2.2.2 Tuning basé sur les prompts

Les méthodes telles que *prefix-tuning*, *prompt-tuning* ou *p-tuning* [8] utilisent des vecteurs d'entrée spéciaux appelés *prompts* :
— ils sont concaténés à l'entrée ou injectés dans les couches internes,
— seuls ces vecteurs sont optimisés lors du fine-tuning.

Exemple : Prefix-tuning

Un préfixe appris est injecté dans les clés et valeurs du mécanisme d'attention à chaque couche du modèle, sous forme de représentations entraînables et fixes.

2.2.3 Fine-tuning partiel ou sélectif

Certaines méthodes PEFT ne mettent à jour que des parties spécifiques du modèle, sélectionnées selon :
— la structure interne (ex. attention vs feedforward),
— la proximité des couches avec la sortie (fine-tuning en surface),
— des critères basés sur les gradients ou l'impact des modifications de poids.

2.2.4 Entraînement peu coûteux

— Le nombre de paramètres entraînés est très faible, généralement entre **0,1% et 3%** du modèle complet.
— Seuls ces paramètres calculent des gradients.

— Le modèle peut souvent être entraîné sur un seul GPU grand public, comme une carte **RTX**.
— Par exemple, QLoRA peut être entraîné avec seulement **24GB** de VRAM, voire parfois entre **8 et 12GB**.

Chapitre 3

Les jeux de données pour la détection automatique de l'inférence textuelle et des contradictions entre phrases en français

L'inférence textuelle vise à déterminer si une hypothèse H peut être considérée comme **déduite**, **contradictoire** ou **sans lien clair** à partir d'un texte T. Elle joue un rôle clé dans plusieurs domaines du TALN comme la recherche d'information, le *question answering* ou encore la vérification de faits. Exemple :
prémisse : La femme porte un manteau rouge et marche sous la pluie.
Hypothèse : Trois garçons sautent dans les feuilles
étiquettes :*Entailment*

Les approches modernes, souvent basées sur les architectures *Transformers*, montrent des limites en matière de robustesse sémantique et dépendent fortement des données sur lesquelles elles ont été entraînées.

Dans cette optique, cinq corpus en français ont été sélectionnés pour évaluer les performances des modèles d'inférence, qu'ils soient multilingues ou spécialisés : DACCORD, RTE3-FR, GQNLI-FR , FraCaS, SICK.

3.1 FraCaS

Le corpus FraCaS (Framework for Computational Semantics), initialement conçu en anglais par des spécialistes en sémantique, a été traduit en français dans le cadre de travaux antérieurs. Il se distingue par sa forte structuration logique et ses paires construites manuellement. Bien que limité en volume, il présente des exemples précis et rigoureux. Chaque paire est annotée avec l'une des trois étiquettes : entailment, contradiction ou neutral.

3.2 DACCORD

Corpus français Il est centré sur les contradictions, avec des phrases longues et complexes, au moment où le reste des jeux de données n'ont qu'entre 9 et 14% des contradictions. , appliqués à des thématiques sensibles comme la guerre en Ukraine, la pandémie de Covid-19 ou la crise climatique. Il contient 1034 paires, construites à la main à partir d'articles, souvent modifiées de manière légère. Environ 49% des exemples relèvent de la contradiction. Chaque paire est annotée comme contradiction ou compatible, sans mention explicite de la véracité.

3.3 RTE3-FR

Version française du corpus RTE3, traduite manuellement avec un appui sur les versions italienne et allemande afin de garantir la précision. Les annotations suivent les trois catégories classiques de la reconnaissance d'inférence textuelle : entailment, neutral et contradiction.

3.4 SICK-Fr

Le SICK-Fr est une version française du dataset SICK. Elle est obtenue généralement par traduction automatique ou manuelle Il a été créé à partir de phrases extraites de deux jeux de données existants : des descriptions d'images (ImageFlickr) et des descriptions vidéo (SemEval 2012 STS MSRVideo).

3.5 GQNLI-FR

Corpus dérivé de Google QA NLI. GQNLI-FR est une traduction de GQNLI, un jeu de données dont les prémisses ont été prises de SNLI et ANLI, les hypothèses ont été construites à l'aide d'un RoBERTa fine-tuné sur MNLI, et les annotations ont été faites par deux de ses auteurs.

		Nombre d'exemples	**implication**	**Neutre**	**Contradiction**
SICK-fr	Train	4439	1274	2524	641
	Validation	495	143	281	71
	Test	4906	1404	2790	712
RTE3	Test	800	409	318	73
	Validation	800	409	300	91
FraCaS	Train	346	204	98	33
GQNLI	Test	300	97	100	103

TABLE 3.1 – Répartition des exemples par classe dans les jeux de données utilisés

Chapitre 4

Performances des modèles

4.1 Les modèles de langage pré-entraîné

Ces dernières années, les modèles de langage pré-entraînés (PLM) ont complètement changé la donne dans le domaine du traitement automatique du langage naturel . Ils ont permis d'atteindre des résultats impressionnants sur plein de tâches comme la traduction, la reconnaissance d'entités nommées ou encore la génération de texte. Leur efficacité repose principalement sur l'architecture Transformer, apparue en 2017, qui utilise un mécanisme d'attention pour mieux comprendre le contexte dans un texte.

Le modèle BERT, par exemple, a joué un rôle clé dans cette révolution. Il a introduit l'idée d'un pré-entraînement non supervisé sur de très grands corpus, avec un objectif simple : prédire des mots masqués en tenant compte du contexte dans les deux sens (avant et après le mot). À partir de ce principe, plusieurs variantes spécialisées ont été créées pour répondre à des besoins précis, notamment dans le domaine médical, avec des modèles comme BioBERT ou ClinicalBERT, qui ont été entraînés sur des textes biomédicaux. Certains, comme PubMedBERT, ont même été entraînés entièrement depuis zéro, uniquement sur des données de santé.

Du côté du français, on retrouve des modèles comme CamemBERT,CamemBERTa, FlauBERT ou encore DrBERTa, qui ont été développés pour mieux gérer les spécificités de notre langue. Et pour répondre à des contraintes techniques (par exemple, pour une utilisation sur mobile ou avec peu de mémoire), d'autres versions plus légères comme DistilBERT ou TinyBERT ont vu le jour. Sans oublier les modèles multilingues comme mBERT ou XLM-R, qui sont utiles lorsqu'on travaille sur plusieurs langues à la fois ou sur des langues peu représentées.

Mais ce qui rend ces modèles intéressants, c'est qu'on peut les adapter à presque n'importe quelle tâche grâce à une phase de *fine-tuning*. Cette étape consiste à prendre le modèle préentraîné et à le spécialiser sur une tâche bien précise, même avec peu de données. C'est ce qui rend les PLM aussi puissants et flexibles.[10]

4.2 Evaluations et resultats initiales

dans le cadre des ont évalué des modèles de pointe basés sur des Transformers (DistilmBERT, XLM-R, mDeBERTa-v3, CamemBERT) sur trois jeux de données français

(DACCORD, RTE3-FR, GQNLI-FR). Ces modèles sont partiellement ou totalement entraînés sur des données françaises. Les auteurs ont évalué plusieurs modèles multilingues sur différentes versions linguistiques du jeu de données RTE3 (anglais, italien, allemand, français) pour une tâche d'inférence textuelle à trois étiquettes. Les résultats, présentés dans deux tableaux, montrent que les performances sur les versions françaises sont cohérentes avec celles obtenues sur les versions dans d'autres langues, ce qui confirme la robustesse des modèles multilingues et monolingues pour la tâche d'inférence textuelle à trois classes.

Modèles	RTE3-EN		RTE3-FR		GQNLI		GQNLI-FR	
	Accuracy	F1	*Accuracy*	F1	*Accuracy*	F1	*Accuracy*	F1
DistilmBERT$_{\text{Base-cased}}$	60,75	47,92	61,13	46,65	26,00	26,03	27,67	26,88
XLM-R$_{\text{Base}}$	-	-	60,50	49,61	-	-	31,67	31,46
CamemBERT$_{\text{Base, 3-class}}$	-	-	63,13	51,52	-	-	33,67	33,44
mDeBERTa-v3$_{\text{Base, XNLI}}$	67,13	56,26	67,13	55,01	28,33	27,73	28,67	27,94
mDeBERTa-v3$_{\text{Base, NLI-2mil7}}$	71,25	61,33	69,63	60,57	**36,67**	**37,04**	**38,33**	**38,58**
XLM-R$_{\text{Large}}$	**72,88**	**63,62**	**71,25**	**62,47**	35,33	35,02	36,34	35,94
CamemBERT$_{\text{Large, 3-class}}$	-	-	71,13	61,97	-	-	33,33	31,62

Figure 4.1 – Evaluation des modèles multilingues sur RTE3 (anglais/original, italien, allemand, français/notre version) pour une tâche NLI incluant des modèles monolingues français

Modèles	DACCORD		XNLI		RTE3-FR		GNLI-FR	
	Accuracy	F1 Score	*Accuracy*	F1 Score	*Accuracy*	F1 Score	*Accuracy*	F1 Score
DistilmBERT$_{\text{Base-cased}}$	63,73	52,59	79,98	68,01	79,63	11,89	51,67	19,89
XLM-R$_{\text{Base}}$	71,57	67,62	87,17	81,14	77,75	21,93	49,33	23,23
CamemBERT$_{\text{Base, 3-class}}$	77,76	76,19	89,64	85,09	80,36	26,29	50,33	**36,05**
mDeBERTa-v3$_{\text{Base, XNLI}}$	80,75	78,30	90,98	86,39	85,75	30,49	52,00	20,88
mDeBERTa-v3$_{\text{Base, NLI-2mil7}}$	80,95	78,47	90,76	85,89	87,00	38,82	50,67	34,51
XLM-R$_{\text{Large}}$	82,01	80,00	**96,49**	**94,74**	86,75	41,11	**53,33**	25,53
CamemBERT$_{\text{Large, 3-class}}$	83,27	81,01	92,30	88,12	**87,63**	**41,42**	52,67	31,07
CamemBERT$_{\text{Large, 2-class}}$	**84,24**	**82,49**	91,70	87,66	85,75	37,36	48,00	19,59

Figure 4.2 – Résultats de la détection des contradiction par Transformers sur DACCORD, XNLI, RTE3-FR et GQNLI-FR

le tableau 4.1 fournit les performances des mêmes modèles, ainsi que de modèles monolingues français, sur RTE3-FR et GQNLI-FR, permettant de comparer plus finement les capacités des modèles sur des corpus en français.

Les résultats mettent en évidence une amélioration progressive des performances entre les différents modèles, notamment entre DistilmBERT et mDeBERTa. Cependant, les modèles préalablement entraînés sur XNLI présentent systématiquement des performances inférieures sur les nouveaux jeux de données (RTE3-FR, DACCORD, GQNLI-FR). Cette diminution s'explique par la complexité accrue de ces datasets, qui ont été spécifiquement conçus pour mettre à l'épreuve les capacités des systèmes existants.

Modèles	DACCORD			RTE3-FR			GNLI-FR		
	Precision	*Recall*	F1	*Precision*	*Recall*	F1	*Precision*	*Recall*	F1
DistilmBERT$_{\text{Base-cased}}$	75,36	40,39	52,59	9,82	15,07	11,89	23,08	17,48	19,89
XLM-R$_{\text{Base}}$	78,12	59,61	67,62	16,13	34,25	21,93	24,21	22,33	23,23
CamemBERT$_{\text{Base, 3-class}}$	81,60	71,46	76,19	20,00	38,36	26,29	**32,31**	**40,78**	**36,05**
mDeBERTa-v3$_{\text{Base, XNLI}}$	89,30	69,71	78,30	27,47	34,25	30,49	24,05	18,45	20,88
mDeBERTa-v3$_{\text{Base, NLI-2mil7}}$	89,75	69,71	78,47	34,02	45,21	38,82	31,71	37,86	34,51
XLM-R$_{\text{Large}}$	89,64	72,23	80,00	34,58	**50,68**	41,11	28,24	23,30	25,53
CamemBERT$_{\text{Large, 3-class}}$	93,18	71,65	81,01	**36,46**	47,95	**41,42**	31,07	31,07	31,07
CamemBERT$_{\text{Large, 2-class}}$	**92,31**	**74,56**	**82,49**	31,20	46,58	37,36	20,88	18,45	19,59

Figure 4.3 – Score F1 détaillé pour la détection d'étiquettes contradictoires par les transformateurs sur DACCORD, RTE3-FR et GQNLI-FR

Afin de mieux évaluer la capacité des modèles à détecter les contradictions, les auteurs ont mesuré spécifiquement l'exactitude et le score F1 associés à la prédiction de l'étiquette « contradiction ». Les résultats de cette analyse ciblée sont présentés dans le tableau 4.2, tandis que le tableau 4.3 détaille les scores de précision et de rappel correspondants. Ces analyses confirment que les modèles les plus performants sur XNLI (notamment XLM-R et CamemBERT) restent globalement les meilleurs sur RTE3-FR et DACCORD, bien que leurs performances y soient inférieures.

Enfin, les performances très faibles observées sur GQNLI, notamment en français, suggèrent que les modèles évalués apprennent davantage des similarités sémantiques superficielles que des relations logiques complexes. Cela s'avère particulièrement problématique dans les cas de contradictions impliquant des quantificateurs ou des chiffres. Les auteurs suggèrent que l'ajout de corpus traduits automatiquement, tels que LingNLI ou SICK, pourrait renforcer les capacités des modèles, et que les jeux de données introduits pourraient être utilisés comme base d'entraînement pour des tâches en zero-shot. [11]

Le tableau récapitulatif 3.1 donne les nombres d'exemple par classe de chaque jeux de données

Deuxième partie

Nos Travaux

4.3 Traitement des données

Dans le cadre de notre projet, nous avons entraîné nos modèles sur plusieurs jeux de données étudiés précédemment. Nous avons d'abord développé un modèle de classification multi-classes à trois étiquettes (*contradiction*, *neutral*, *entailment*), en nous appuyant sur les jeux GQNLI, SICK, RTE3 et FraCaS. Ces jeux de données contiennent des exemples illustrant diverses relations entre phrases. Le corpus FraCaS se distingue notamment par l'importance des quantificateurs généralisés ce qui rend la tâche de classification particulièrement complexe. Par exemple :
Prémisse : Tout Italien veut être un grand ténor. Tout Italien veut être un grand ténor. Certains Italiens sont de grands ténors.
Hypothèse : Il y a des Italiens qui veulent être de grands ténors.
Étiquette : 0

Enfin, pour mieux tester les performances, nous avons créé un deuxième modèle en transformant la tâche en classification binaire, en regroupant les labels en "contradiction" versus "non-contradiction". Pour ce faire, nous avons retravaillé les jeux RT3 et SICK, en modifiant leurs annotations pour coller à ce format binaire.

Dans notre démarche, nous avons aussi cherché à optimiser la quantité d'exemples nécessaires à l'entraînement. Nous avons expérimenté avec des ensembles très petits, allant de 20 à 40 exemples par classe, pour observer jusqu'où le modèle pouvait apprendre avec peu de données. Par ailleurs, nous avons testé différents mélanges de jeux de données, en jouant sur la diversité des exemples et sur la différence à la nature des données de chaque jeu de Données. Par exemple, GQNLI et RTE contiennent des exemples très différents, tandis que SICK et FraCaS proposent des exemples plus proches les uns des autres. Certains de ces mélanges se sont révélés fructueux, tandis que d'autres ont moins bien fonctionné, et nous détaillerons tout cela plus précisément dans la section des résultats.

4.4 Entraînement d'un modèle de classification de texte avec LoRA et PEFT

4.4.1 Le choix du modèle de base

Dans ce projet, nous avons choisi CamemBERTa-v2-base [1], un modèle développé par l'équipe ALMAnaCH (Inria) et basé sur DeBERTaV2, préentraîné sur un vaste corpus de 275 milliards de tokens en français. Il utilise un *tokenizer WordPiece* performant, prend en charge des contextes longs (jusqu'à 1024 tokens) et applique la stratégie *Replaced Token Detection (RTD)*, améliorant la qualité des représentations. Ce modèle offre d'excellents résultats sur plusieurs *benchmarks* francophones tout en restant léger, ce qui le rend adapté à des environnements à ressources limitées.

Notre objectif est de construire un modèle de classification capable de distinguer trois classes, en optimisant les performances sans dépasser nos contraintes matérielles. Faute de GPU puissants, il est trop coûteux de fine-tuner entièrement un grand modèle. Nous avons donc opté pour la méthode LoRA, une approche PEFT qui permet d'entraîner

1. `https://huggingface.co/almanach/camembertav2-base-xnli`

uniquement une petite partie des paramètres, réduisant ainsi la mémoire nécessaire et le temps d'entraînement, tout en maintenant de bonnes performances.

Le choix de `almanach/CamemBERTa-base`, une version francophone de **DebBERTa** qui est lui même basé sur **RoBERTa** , repose sur trois points :

— Il est entraîné sur des données françaises, adaptées à notre corpus.
— Sa taille réduite facilite son utilisation dans un contexte à ressources limitées.
— Il conserve des performances solides, même avec un fine-tuning partiel via LoRA.

2. Application de LoRA et configuration de l'entraînement

Une fois le modèle **CamemBERTa** chargé via `AutoModelForSequenceClassification` (en précisant trois classes), nous avons intégré la méthode **LoRA** [2] pour effectuer un fine-tuning allégé. Pour cela, nous avons défini une configuration avec les paramètres suivants :

```
model = AutoModelForSequenceClassification.from_pretrained(
    model_id,
    num_labels=3,
    device_map="auto"
)
lora_config = LoraConfig(
    r=32,
    lora_alpha=256,
    target_modules=["query_proj", "value_proj"],
    lora_dropout=0.05,
    task_type=TaskType.SEQ_CLS,
)

lora_model = get_peft_model(model, lora_config)
lora_model.print_trainable_parameters()
```

Figure 4.4 – configuration de LoRA

Après avoir testé plusieurs configurations, ces valeurs ont été choisies pour trouver un bon compromis entre *performance* et *régularisation*, et ainsi limiter le risque de sur-apprentissage . Nous avons ciblé les modules `query_proj` et `value_proj`, qui sont des éléments clés du mécanisme d'attention dans les *transformeurs*. En se concentrant sur ces parties, nous capturons les interactions essentielles entre les tokens sans devoir entraîner l'ensemble du modèle.

Nous avons ensuite utilisé la fonction `get_peft_model` pour convertir CamemBERTa en un modèle compatible avec LoRA . Avant de démarrer l'entraînement, nous avons affiché les *paramètres entraînables* afin de vérifier que seule une petite portion du modèle serait effectivement mise à jour.

2. `https://huggingface.co/blog/samuellimabraz/peft-methods`

L'entraînement est géré avec la classe `Trainer` de *HuggingFace*[3], qui facilite grandement la mise en place du processus. Voici les principales configurations adoptées :

— 20 époques,
— une taille de batch de 16,
— une stratégie d'évaluation et de sauvegarde par époque avec `eval_strategy="epoch"` et `save_strategy="epoch"`,
— l'option `load_best_model_at_end=True` activée pour charger automatiquement le meilleur modèle sur le jeu de validation.

Cette configuration nous permet d'assurer un suivi rigoureux des performances, tout en tirant parti des avantages de LoRA dans un environnement limité en ressources.

4.5 Traitement des jeux de données et évaluation des performances

Pour l'apprentissage, nous avons entraîné et validé sur différents jeux de données. Chaque jeu de données est passé par une fonction `preprocess_function` qui s'occupe de la *tokenisation*, du *padding* et du nettoyage textuel. Cette étape est essentielle avant d'envoyer les données au modèle.

```
model_id =  "almanach/camembertav2-base-xnli"
tokenizer = AutoTokenizer.from_pretrained(model_id, use_fast=True)
def preprocess_function(examples):
    encodings = tokenizer(examples['premise'], examples['hypothesis'],
                          padding="max_length", truncation=True)
    encodings['label'] = examples['label']
    return encodings
```

Figure 4.5 – Tokenisation des données

Afin de mieux suivre l'évolution de l'entraînement, nous avons mis en place un callback personnalisé qui évalue les performances du modèle sur le jeu de test à la fin de chaque époque, **sans interférer avec le processus d'apprentissage**. Contrairement à la validation, réalisée sur un sous-ensemble des données d'entraînement et pouvant influencer les ajustements du modèle. cette évaluation sur le jeu de test reste totalement neutre. Le callback utilise la méthode `predict()` pour générer les prédictions, puis calcule plusieurs métriques de classification. Celles-ci sont automatiquement enregistrées et visualisées via **Weights & Biases (WandB)**, permettant ainsi un suivi clair et régulier des performances du modèle à chaque époque. Les métriques de classification :

— **Accuracy** : proportion de prédictions correctes sur l'ensemble des échantillons.

$$\text{Accuracy} = \frac{\text{Nombre de prédictions correctes}}{\text{Nombre total de prédictions}}$$

3. `https://huggingface.co/docs/transformers/main_classes/trainer`

— **Précision** : mesure la proportion de vraies prédictions positives parmi toutes les prédictions positives.

$$\text{Précision} = \frac{\text{VP}}{\text{VP} + \text{FP}}$$

— **Rappel** : mesure la proportion de vraies prédictions positives parmi tous les échantillons réellement positifs.

$$\text{Rappel} = \frac{\text{VP}}{\text{VP} + \text{FN}}$$

— **F1-score macro** : moyenne harmonique entre précision et rappel, calculée pour chaque classe puis moyennée.

$$\text{F1} = 2 \times \frac{\text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}}$$

Ces métriques sont ensuite enregistrées avec **WandB**, ce qui nous permet de suivre les résultats en temps réel.

Chapitre 5

Expériences et évaluations

5.1 Modèle pour l'inférence textuelle à trois étiquettes

5.1.1 Pré-évaluation des jeux de données sur le modèle préentraîné CamemBERTa_base_XNLI

Le tableau 5.1 présente les résultats de l'évaluation des jeux de données sur le modèle `CamemBERTa_base_XNLI`, sans étape de fine-tuning préalable.

modèle	**FraCaS**		**SICK**		**RTE3**		**GONLI**	
	Acc	F1	Acc	F1	Acc	F1	Acc	F1
CamemBERTa-xnli	0.6325	0.6621	0.5664	0.5617	0.5653	0.6662	0.2483	0.2733

TABLE 5.1 – Résultats des performances du modèle CamemBERTa_base_XNLI sur différents jeux de données

5.1.2 Résultats

Entraînement	**Validation**	**FraCaS**		**SICK**		**RTE3**		**GONLI**	
		Acc	F1	Acc	F1	Acc	F1	Acc	F1
GQ + SICK	GQ + SICK	0.70	0.44	0.73	0.72	0.72	0.59	0.38	0.38
rte3	rte3	0.73	0.47	0.57	0.58	0.71	0.60	0.33	0.32
SICK	SICK	0.74	0.5	0.73	0.72	0.72	0.60	0.40	0.39
SICK + fracs	SICK	0.52	0.35	0.67	0.60	0.63	0.55	0.40	0.36
SICK	SICK + FraCaS	0.67	0.42	0.69	0.68	0.72	0.59	0.35	0.32
GQ	GQ + FraCaS	0.67	0.43	0.61	0.62	0.69	0.59	0.28	0.27
GQ + FraCaS	GQ	0.64	0.42	0.58	0.59	0.61	0.55	0.50	0.42
GQ	GQ	0.67	0.42	0.62	0.62	0.70	0.62	0.28	0.26
GQ + SICK	GQ + SICK + FraCaS	0.70	0.44	0.70	0.68	0.72	0.60	0.40	0.40

TABLE 5.2 – Résultats de performance des différents jeux de données sur notre modèle

5.1.3 Analyse des Résultats par Jeu de Donnée

L'analyse des performances selon les combinaisons d'entraînement et de validation permet de tirer plusieurs constats intéressants.

FraCaS

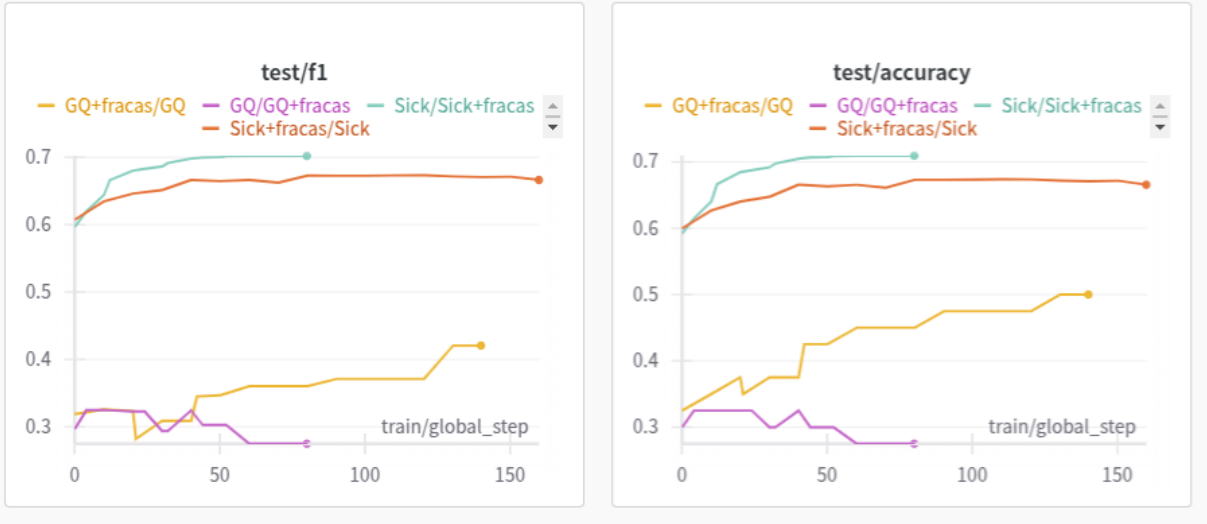

Figure 5.1 – Evaluations des diffèrents jeu de Données qui utilise FraCaS

Les résultats sur FraCaS restent globalement faibles, avec un score F1 maximal de 0.68 , constater du tableau 5.2 atteint lorsque l'on utilise comme jeu de validation uniquement SICK. Cela s'explique en partie par la nature même de FraCaS : c'est un petit jeu de données, déséquilibré, comprenant seulement 346 exemples. Les 74 premiers se concentrent spécifiquement sur les phénomènes liés aux quantificateurs généralisés, qui sont précisément ceux abordés dans le corpus GQNLI, avec des phrases souvent plus complexes que celles de FraCaS. Les exemples restants du corpus FraCaS couvrent d'autres phénomènes linguistiques intéressants. Bien qu'ils ne soient pas forcément pertinents pour des tâches centrées sur les quantificateurs généralisés, comme celles de GQNLI, ils peuvent tout à fait être utilisés pour d'autres types d'expériences, que ce soit pour l'entraînement ou l'évaluation de modèles.

FraCaS a un effet positif lorsqu'il est intégré à l'entraînement avec GQNLI, comme on peut le voir dans la figure 5.1 car il introduit une variété linguistique et des formulations différentes autour des quantificateurs généralisés, ce qui peut enrichir la capacité du modèle à généraliser. Cependant, cet effet reste limité à cause de la quantité réduite d'exemples, ce qui rend l'impact peu significatif en termes de variation globale des performances.

Par ailleurs, FraCaS a également un impact positif dans la validation avec SICK, car les structures logiques présentes dans FraCaS, notamment celles liées à l'inférence à partir de quantificateurs et de constructions complexes, aident le modèle à mieux capturer certaines régularités logiques.

RTE3

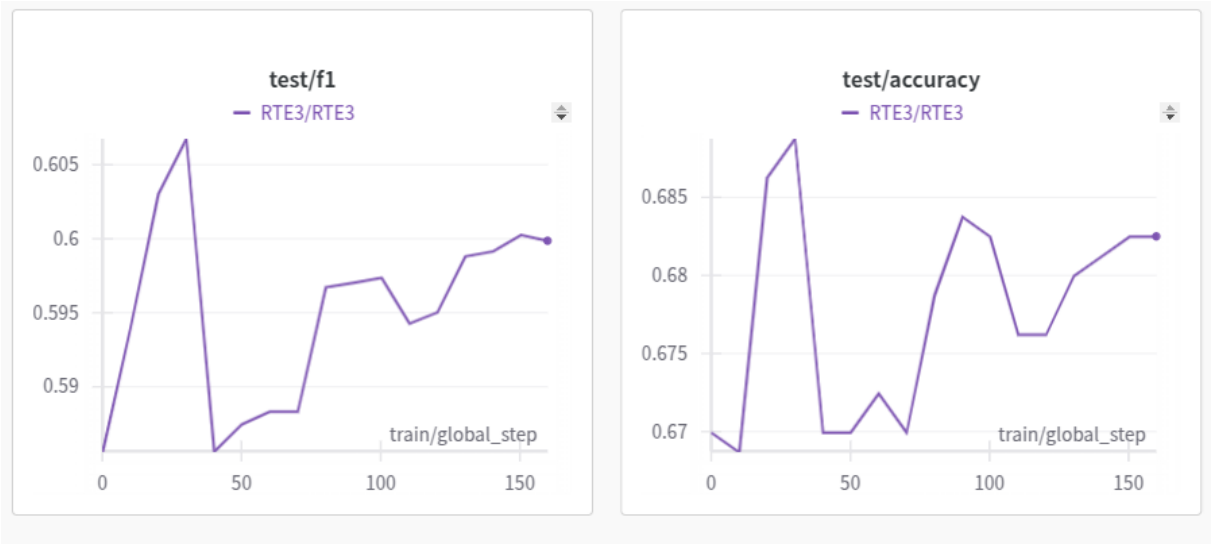

Figure 5.2 – Résultats sur RTE3

Concernant RTE3, les résultats sont étonnamment faibles et stagnants, traduisant un *underfitting* persistant. Ce comportement semble lié aux limitations propres à LoRA, qui n'adapte qu'une portion restreinte des paramètres du modèle. Or, les exemples de RTE3 sont souvent longs, syntaxiquement complexes, et nécessitent une capacité d'adaptation plus fine pour capturer les subtilités logiques. Dans un contexte few-shot, cette faible plasticité induite par LoRA empêche le modèle d'apprendre des représentations suffisamment riches et nuancées, ce qui expliquerait pourquoi les performances ne progressent pas malgré l'entraînement.

SICK

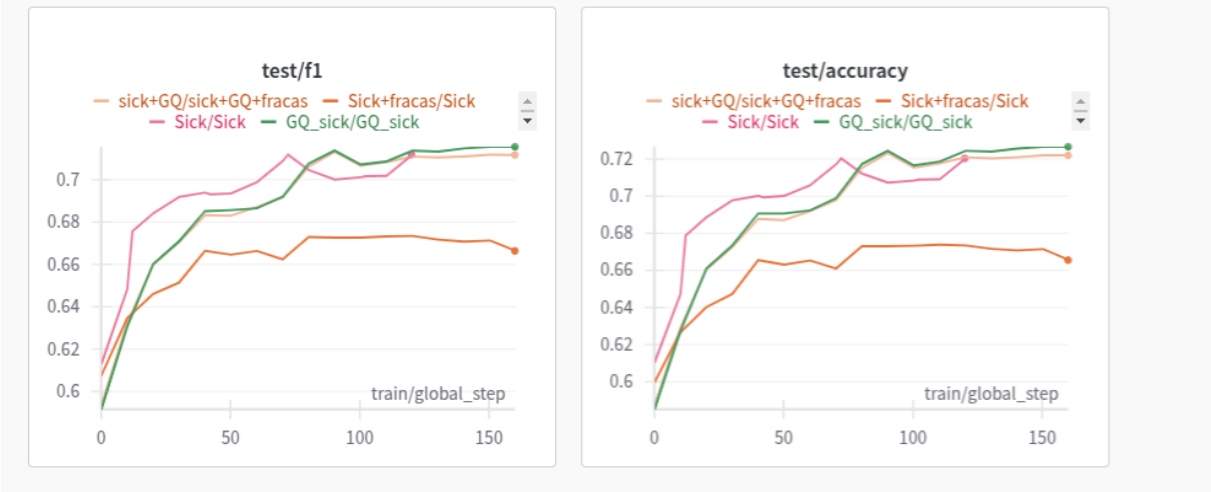

Figure 5.3 – Résultats sur SICK

SICK affiche d'excellents résultats dans presque tous les cas, même avec un nombre très restreint d'exemples. Ainsi, **20 exemples par classe** ont suffi pour obtenir des performances significatives, comme le montre la figure 5.3. Le modèle atteint des scores F1 supérieurs à 0.71 5.2 lorsqu'il est entraîné uniquement sur SICK, ou en combinaison avec GQ, voire même lorsqu'il est évalué sur SICK après un entraînement préalable sur SICK et valider sur SICK + FraCaS. Cela suggère que ce jeu est particulièrement bien adapté au *few-shot learning*, grâce à sa structure homogène, ses formulations claires et son faible niveau de bruit. Ces caractéristiques en font une base solide pour l'apprentissage des relations textuelles et favorisent la généralisation vers d'autres jeux.

Les matrices de confusion obtenues sur SICK 5.10 seul ainsi que sur SICK et validation SICK+FraCaS 5.12 confirment cette stabilité : le modèle parvient à bien distinguer les différentes classes, avec peu de confusions entre les relations.

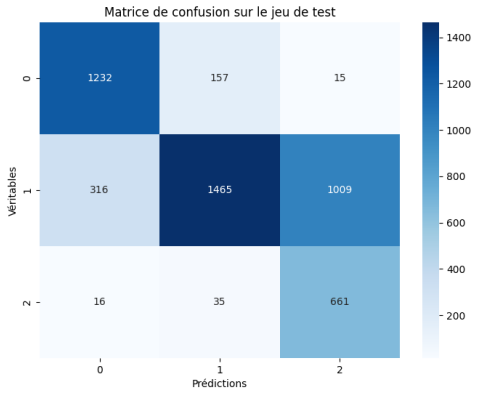

Figure 5.4 – Matrice de confusion évaluation sur SICK de SICK / SICK+FraCaS

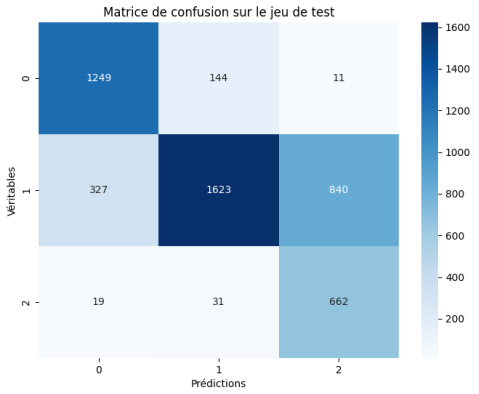

Figure 5.5 – Matrice de confusion évaluation sur SICK de SICK

GQNLI

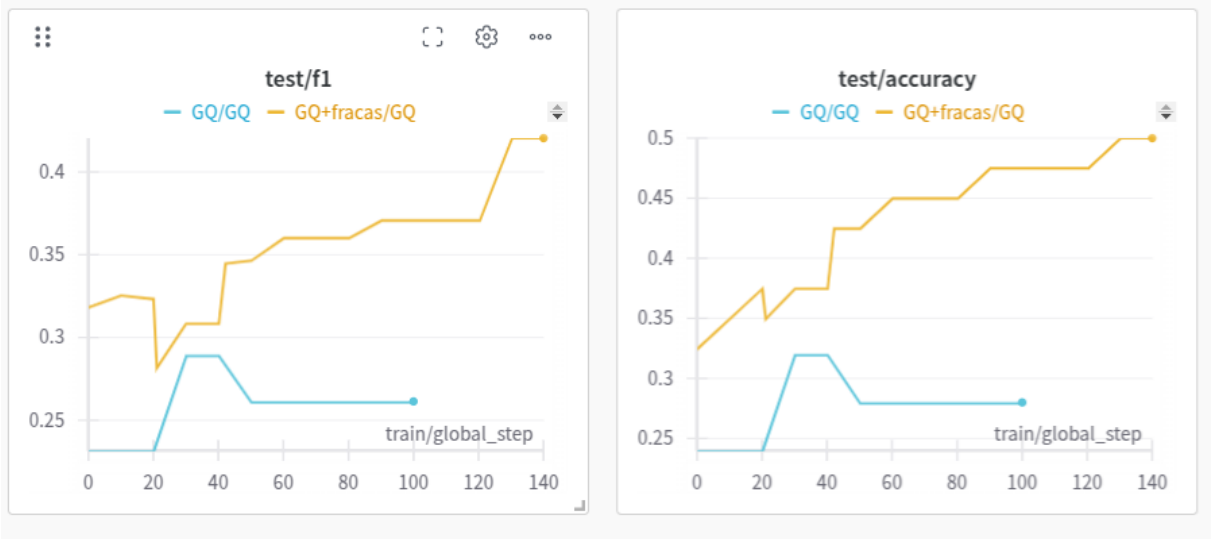

Figure 5.6 – Résultats sur GQNLI

GQNLI présente un comportement particulier. Malgré ses 300 exemples, les performances restent modestes, avec une *Accuracy* maximale autour de 0,50 et un F1 score de 0.42 5.6. Cela s'explique par deux facteurs principaux :

D'abord, la variabilité sémantique des exemples est très faible : de nombreuses paires se ressemblent fortement (par exemple : « il y a six chiens » / « il y a six chats »), ce qui rend difficile l'apprentissage de distinctions fines par le modèle.

Ensuite, seulement 20 exemples par classe ont été utilisés pour l'entraînement, 40 pour l'évaluation , et les 200 restants pour la validation. Ce volume réduit rend l'apprentissage d'autant plus difficile. L'association avec FraCaS a permis une légère amélioration, mais cet effet reste limité : FraCaS ne contient que 75 exemples sur les quantificateurs généralisés, avec très peu de contradictions, ce qui crée un fort déséquilibre entre les classes. Ainsi, les contradictions, moins bien représentées dans l'ensemble global, sont aussi moins bien apprises et généralisées.

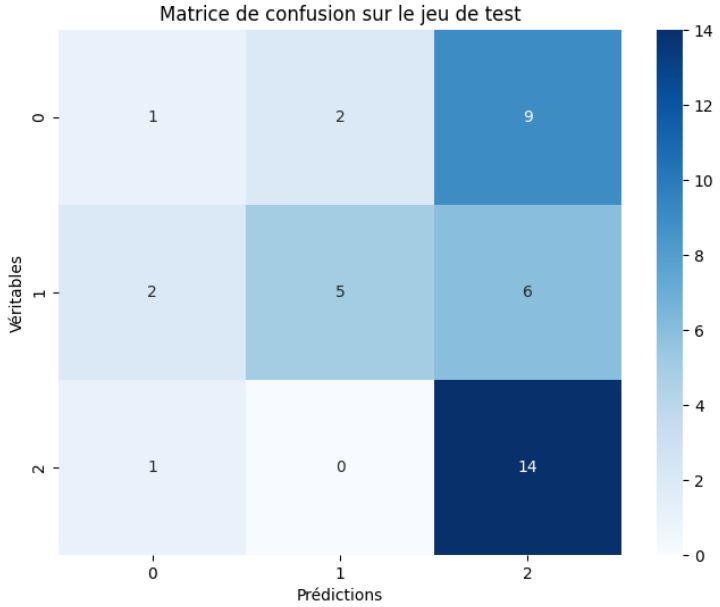

Figure 5.7 – Matrice de confusion de GQNLI + FraCaS dur GQNLI

Malgré les contraintes, les résultats obtenus sur GQNLI restent acceptables, comme le montre la matrice de confusion. Ils pourraient toutefois être améliorés en utilisant un modèle plus performant, tel que LLAMA ou mDeBERTa. Par ailleurs, un entraînement conjoint avec un corpus comme RTE3 pourrait avoir un impact positif et améliorer significativement les performances du modèle.

5.2 Modèle pour l'inférence textuelle a deux étiquette

5.2.1 Pré-évaluation des jeux de données sur le modèle préentraîné CamemBERTa_base_XNLI

Le tableau 5.3 présente les résultats de l'évaluation des jeux de données après adaptation sur le modèle `CamemBERTa_base_XNLI`, sans étape de fine-tuning préalable.

Modèle	**SICK**		**RTE3**		**DACCORD**	
	Acc	**F1**	**Acc**	**F1**	**Acc**	**F1**
CamemBERTa_base_XNLI	0.36	0.20	0.59	0.2762	0.46	0.21

Table 5.3 – Résultats des performances du modèle CamemBERTa_base_XNLI sur différents jeux de données

5.2.2 Discussion

Nombre minimal d'exemples nécessaires pour entraîner le modèle (classification binaire)

Dans le cadre d'une tâche de classification binaire avec les étiquettes contradiction et non-contradiction, les premiers tests indiquent que 20 exemples par classe représentent un minimum nécessaire pour que le modèle commence à apprendre des motifs pertinents. Bien que les performances à ce stade restent limitées, elles montrent une tendance à l'amélioration à mesure que le nombre d'exemples augmente. une augmentation de 10 a 20% dans le score *Accuracy* est observée entre 30 et 50 exemples par classe, plage dans laquelle le modèle parvient au résultats optimal . En revanche, au-delà de 60 exemples par classe, une dégradation des performances a été constatée, probablement due à un surapprentissage (*overfitting*) . Ainsi, un équilibre semble se situer entre 30 et 50 exemples par classe pour obtenir de bons résultats dans ce contexte binaire.

5.2.3 Analyse des résultats sur le modèle

Train/val	**Modèle**	**Daccord**		**SICK**		**RTE3**	
		ACC	**F1**	**ACC**	**F1**	**ACC**	**F1**
SICK/SICK		0.6850	0.4065	0.8486	0.5172	0.7312	0.5078
daccord/daccord	CamemBERTa	0.7350	0.4334	0.50	0.4192	0.7125	0.3391
rte3/rte3		0.54	0.2237	0.40	0.25	0.5887	0.2822

Table 5.4 – Résultats de classification (ACC et F1) sur les jeux de données Daccord, SICK et RTE3

Figure 5.8 – Graphe des Validation (Acc/epoch et F1/epoch) sur le modèle

Figure 5.9 – Graphe des Testes (Acc/epoch et F1/epoch) sur le modèle

5.2.4 Résultat et Analyse

Dans le cadre du *few-shot learning*, les performances obtenues demeurent globalement modestes, notamment sur les jeux de données SICK et DACCORD. En comparaison avec les résultats du modèle CamemBERTa non fine-tunè (cf. tableau 5.3), nos scores présentent néanmoins une amélioration générale, malgré un entraînement à 40 exemples par classe. Ceci illustre l'efficacité de notre approche légère reposant sur LoRA pour l'adaptation de modèles pré-entraînés.

Pour les *datasets* SICK et DACCORD, les scores F1 obtenus surpassent ceux du modèle pré-entraîné seul, démontrant ainsi la capacité de l'adaptation par LoRA à affiner le modèle avec un nombre restreint d'exemples annotés.

À l'inverse, les performances sur RTE3 sont nettement inférieures. Cette dégradation semble s'expliquer par les limitations inhérentes à LoRA, qui n'adapte qu'un sous-

ensemble restreint des paramètres du modèle. Or, les exemples du corpus RTE3 se caractérisent souvent par des phrases longues et syntaxiquement complexes, requérant une capacité d'adaptation plus fine afin de saisir les subtilités logiques présentes.

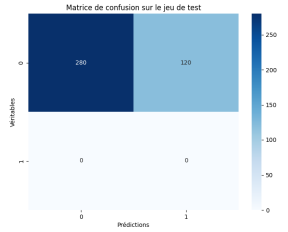

Figure 5.10 – Matrice de confusion évaluation de DACCORD classification binaire

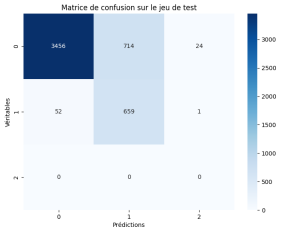

Figure 5.11 – Matrice de confusion évaluation de SICK classification binaire

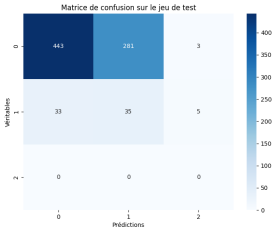

Figure 5.12 – Matrice de confusion évaluation de RTE classification binaire

Chapitre 6

Conclusion et perspectives

6.1 Conclusion sur nos travaux

Notre approche légère basée sur LoRA s'est révélée efficace pour adapter un modèle préentraîné dans un contexte few-shot, même avec seulement 20 à 40 exemples par classe. Les résultats sont particulièrement encourageants sur les jeux de données SICK et DACCORD, dépassant ceux d'un modèle non adapté. L'analyse montre que la structure, la qualité des données et l'équilibre des classes influencent fortement la performance. Si certains jeux comme FraCaS ou GQNLI posent davantage de difficultés, des pistes d'amélioration existent via l'enrichissement des données ou l'usage de modèles plus puissants. Enfin, un nombre modéré d'exemples (30 à 50) semble suffisant pour obtenir de bons résultats tout en évitant le surapprentissage. Ces conclusions ouvrent des perspectives concrètes pour des contextes où les données annotées sont limitées

6.2 Perspective

Au-delà de l'adaptation par LoRA, d'autres techniques de fine-tuning efficaces et économes en ressources méritent d'être explorées dans des contextes few-shot. Parmi elles, le prompt-tuning apparaît comme une alternative prometteuse. Cette méthode consiste à apprendre un vecteur d'entrée optimisé (souvent appelé soft prompt) que l'on insère dans le modèle préentraîné, sans modifier ses poids internes. Elle permet de spécialiser un modèle à une tâche spécifique avec un très faible nombre de paramètres ajustables, ce qui en fait une solution particulièrement adaptée lorsque les ressources en données et en calcul sont limitées.

Une autre approche intéressante est le prefix-tuning, qui introduit un préfixe de vecteurs entraînables au début des couches d'attention du modèle. Comme le prompt-tuning, cette méthode préserve l'intégrité du modèle préentraîné tout en permettant une personnalisation efficace et ciblée sur de nouvelles tâches. Ces stratégies peuvent être particulièrement utiles pour l'inférence textuelle, où il est important de capturer des relations sémantiques fines avec peu d'exemples annotés.

Enfin, l'utilisation de grands modèles de langage comme LLaMA, Mistral ou ChatGPT, via du prompt-tuning ou de l'in-context learning, ouvre également des perspectives intéressantes. Ces modèles sont capables de raisonner à partir d'instructions textuelles et

d'exemples fournis dans le prompt, sans nécessiter de réentraînement complet. Cela permettrait d'évaluer leur capacité à généraliser à des tâches complexes comme l'inférence textuelle en français, tout en réduisant considérablement le besoin d'annotations supervisées.

Bibliographie

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[2] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE : A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018.

[3] Sinong Wang, Han Fang, Madian Khabsa, Hanzi Mao, and Hao Ma. Entailment as few-shot learner, 2021.

[4] Robert Kraig Helmeczi, Mucahit Cevik, and Savas Yıldırım. Few-shot learning for sentence pair classification and its applications in software engineering, 2023.

[5] Lewis Tunstall, Nils Reimers, Unso Eun Seo Jo, Luke Bates, Daniel Korat, Moshe Wasserblat, and Oren Pereg. Efficient few-shot learning without prompts, 2022.

[6] Lydia Abady, Jun Wang, Benedetta Tondi, and Mauro Barni. A siamese-based verification system for open-set architecture attribution of synthetic images, 2023.

[7] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network, 2018.

[8] Samuel Lima Braz. Peft : Parameter-efficient fine-tuning methods for llms, 2025. Consulté le 26 mai 2025.

[9] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora : Low-rank adaptation of large language models. *CoRR*, abs/2106.09685, 2021.

[10] Wissam Antoun, Francis Kulumba, Rian Touchent, Éric de la Clergerie, Benoît Sagot, and Djamé Seddah. Camembert 2.0 : A smarter french language model aged to perfection, 2024.

[11] Maximos Skandalis, Richard Moot, Christian Retoré, and Simon Robillard. New datasets for automatic detection of textual entailment and of contradictions between sentences in French. In Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue, editors, *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 12173–12186, Torino, Italia, May 2024. ELRA and ICCL.