



Compte rendu du case study n°2 :

Conception et déploiement d'un module IA de réponse aux commentaires clients, détection de mots clés & identification de l'aspect d'excuse.

Proposé par : Waad Toumi

Réalisé par : Aymen Fourati

Cadre : Processus de recrutement chez Malou

Date de remise : 27-12-2023

lien vers l'application : <http://20.123.99.158:8080>

Github : <https://github.com/aymenfourati/malin-keyword-sorry-aspect-detection>

Image sur docker hub : kaiken26/projet-devops:mlops-pipeline-for-llm-testing-2.0

Plan

A. Mise en context

B. Conception du module IA

a. Système de réponse aux commentaires clients (Case study n°1)

- i. Besoin et spécifications
- ii. Choix et stack technique
- iii. Préparation des données
- iv. Evaluation
- v. Déploiement

b. Système de détection de mots clés

- i. Besoin et spécifications
- ii. Choix et stack technique
- iii. Evaluation

c. Système de classification de l'aspect d'excuse

- i. Besoin et spécifications
- ii. Choix et stack technique
- iii. Méthode 1 : Rule based
 - 1. Modélisation
 - 2. Evaluation
- iv. Méthode 2 : Classifier binaire LSTM
 - 1. Préparation des données
 - 2. Evaluation
 - 3. Déploiement

C. Interface utilisateur de l'application

D. Déploiement du module IA

Mise en context

Le présent compte rendu est la documentation du projet qui constitue le deuxième case study élaboré dans le cadre du processus de recrutement chez Malou, sous la supervision de Waad Toumi et exécuté par Aymen Fourati. Cette étude se concentre sur la continuation du premier case study qui a traité la conception d'un module d'intelligence artificielle dédié à la réponse aux commentaires clients pour y ajouter une partie de déploiement et des nouvelles fonctionnalités notamment la détection de mots clés et l'identification de l'aspect d'excuse.

Conception du module IA

a. Système de réponse aux commentaires clients (Case study n°1)

i. Besoin et spécifications

Durant le case study numéro 1, on s'est proposé de créer un système de réponse automatiques à des commentaires clients.

ii. Choix et stack technique

Pour des soucis de temps nous avons fait le choix de nous focaliser sur les feedbacks clients dans le domaine de la restauration puisque cela constitue l'expertise de Malou.

Pour ce faire, nous avons tenté de faire le fine tuning de Falcon 7b : un LLM open source et ouvert à l'utilisation commerciale mais le modèle a montré des signes d'instabilité lors de l'entraînement.

Par conséquent, nous avons choisi d'effectuer le fine tuning du LLM gpt2 ; le modèle open source d'open AI. Nous avons pensé qu'il serait judicieux de nous orienter vers leurs technologies pour une éventuelle migration convenable vers un fine-tuned gpt4.

Nous avons aussi utilisé Langchain : un framework d'intégration de LLMs (restant sur la même optique de flexibilité puisque Langchain présente une grande panoplie de sources de LLMs comme openAi) pour intégrer notre modèle pré entraîné dans notre application. Langchain nous a permis de nous connecter à Hugging Face pour consommer l'API de notre modèle sans avoir à télécharger toutes les dépendances en local.

Cette configuration nous permet de passer d'un modèle à un autre sans avoir à effectuer des changements dans l'application web et de garder un pipeline de test du modèle séparé de l'environnement de développement de l'application web.

iii. Préparation des données

Pour ce qui est de la base de données, nous nous sommes basés sur une base de données de commentaires clients trouvés sur Kaggle. Cette base de données de base contient 977 lignes et deux columns : La revue et une classification binaire si la revue est positive ou négative.

Voici quelques lignes de cette base de données :

Customer Reviews	Whether they liked food or not.
977 unique values	705 total values
prompt.	
Would not go back.	0
The cashier had no care what so ever on what I had to say it still ended up being wayyy overpriced.	0
I tried the Cape Cod ravioli, chicken, with cranberry...mmm!	1
I was disgusted because I was pretty sure that was human hair.	0

Nous avons utilisé 340 lignes de cette base de données en passant par chat gpt pour y ajouter deux columns : Une réponse au commentaire et un tableau de mots clés pour contextualiser le commentaire.

Voici quelques lignes de la base de données résultante :

```
{
  "keywords": ["restaurant", "won't be disappointed", "positive experience"],
  "comment": "You won't be disappointed.",
  "reply": "We're glad you feel that way! Your positive feedback about our offerings at the restaurant is appreciated, and we look forward to providing more satisfying experiences.",
  "satisfaction": 1
},
{
  "keywords": ["restaurant", "serve with vinaigrette", "better overall dish", "still very good", "positive experience"],
  "comment": "They could serve it with just the vinaigrette and it may make for a better overall dish, but it was still very good.",
  "reply": "We're pleased to hear you found our dish still very good at the restaurant! Your feedback on serving with vinaigrette is noted, and we appreciate your valuable input.",
  "satisfaction": 1
},
{
  "keywords": ["restaurant", "too many places", "1 egg breakfast", "$4.00", "negative experience"],
```

```
"comment": "I go to far too many places and I've never seen any restaurant  
that serves a 1 egg breakfast, especially for $4.00.",  
"reply": "We apologize for falling short of expectations with our 1 egg  
breakfast pricing at the restaurant. Your feedback is important, and we're  
actively working to address this issue. Thank you for bringing it to our  
attention.",  
"satisfaction": 0  
},
```

Pour des soucis de disponibilités de données, nous n'avons pas pu trouver une solution qui marche sur la langue française mais pour des améliorations futures on pourra éventuellement créer une base de données en traduisant celle ci à partir de la langue anglaise.

En utilisant la librairie transformers de hugging face, nous avons accès aux fonctions de openai optimiser pour leurs model tel que gpt2. Nous utilisons la fonction de tokenization prédéfinie sur notre base de données.

iv. Evaluation

Pour évaluer ce modèle, nous avons monitoré principalement deux KPIs : Le taux de réponses et temps de réponses.

Le temps de réponse est l'une des metrics les plus importantes et qui nous a obligé à nous retourner vers gpt2 au lieu de falcon 7b est le temps de réponse. En effet, dans un environnement prod, on aimerait bien que le client reçoivent une réponse sur son commentaire instantanément sans avoir à quitter l'interface sur lequel il opère, par exemple si le chatbot agit sur les commentaires des posts facebook d'un restaurant. Pour le moment, le test prolongé du modèle pour des entrées variées (plus de 50 essais) nous indique que le temps de réponse est **aux alentours de 20 secondes (min = 11.2 s, max = 35 s)**

Pour ce qui est du taux de réponse, nous avons constaté qu'environ **90% des cas, le système arrivait à délivrer une réponse.**

v. Déploiement

Le modèle a été hébergé sur Hugging Face et tous les appels à partir de l'application on été effectué à travers un API ce qui fait malheureusement accroître le temps de réponse du modèle.

b. Système de détection de mots clés

i. Besoin et spécifications

On se propose maintenant d'évaluer notre modèle de réponse en déterminant les mots clés qu'il a réussi à exploiter dans la réponse au feedback client. Ce système devra répondre à des contraintes de robustesse et de scalabilité. Ainsi que de pouvoir identifier les mots malgré les variations qui peuvent acquérir dû au contexte (pluriel, singulier, féminin, masculin, conjugaison, adverbe.. etc).

ii. Choix et stack technique

Pour ce faire, nous allons implémenter plusieurs techniques de prétraitement afin de pouvoir identifier la présence de ces mots clés.

D'abord nous supprimerons toute forme de liaison lorsqu'il s'agit de la langue française, par exemple la chaîne "d'avoir" devient "avoir".

Ensuite, nous utilisons la technique de stemming qui consiste en la transformation des mots à leurs formes racine pour répondre à la contrainte de variations des formes de chaque mot clés, suivie d'une suppression des stops words, une tokenization et la suppression de la ponctuation et la mise en minuscules pour uniformiser le plus possible les mots à traiter.

Enfin, nous calculons le TF-IDF de chaque mot et à l'aide du calcul de similarité nous pouvons déterminer, si celle-ci excède un certain seuil, la présence d'un mot clé dans une réponse donnée.

La raison pour laquelle nous n'avons pas opté pour une simple recherche des mots clés dans la réponse après la phase de prétraitement mais aussi une vectorization et un calcul de similarité est que, d'une part, parfois un mot clé composé peut avoir une présence dans la phrase de façon discontinue par exemple le mot clé 'authenticité culinaire' dans le cas suivant :

```
Response = "Bienvenue dans notre havre culinaire. Si vous recherchez la  
meilleure pizza de la ville, ne cherchez pas plus loin. Notre restaurant  
italien est ouvert tous les dimanches pour satisfaire vos papilles. Venez  
goûter l'authenticité de nos recettes."
```

```
keywords = ['pizza sur paris', 'authenticité culinaire', 'meilleure pizza',  
'ouvert dimanche', 'havre culinaire', 'recettes authentiques']
```

```
-----  
Output : ['authenticité culinaire', 'meilleure pizza', 'ouvert dimanche',
```



```
'havre culinaire', 'recettes authentiques']
```

Une simple recherche n'aurait pas identifié la présence de ce mot clé.

D'une autre part, ça nous permet d'éviter les faux positif comme dans l'exemple suivant lorsqu'il s'agit du mot clé 'dimanche gourmand' :

```
response = " Nous tenons à vous présenter nos excuses pour tout désagrément lors  
de votre récente visite. Chez nous, chaque dimanche est une opportunité de  
savourer les délices de notre cuisine italienne, y compris nos fameux tiramisus.  
Nous nous efforçons de vous offrir une expérience exceptionnelle."  
keywords = ['service rapide', 'cuisine italienne', 'excuses', 'dimanche  
gourmand', 'tiramisus', 'expérience exceptionnelle']  
-----  
output : ['cuisine italienne', 'expérience exceptionnelle']
```

Nous avons aussi implémenté une librairie de détection de langue pour automatiser le processus de prétraitement suivant la langue de l'input puisque les algorithmes de stemming diffèrent d'une langue à une autre aussi, intuitivement, il ne s'agit pas du même dictionnaire de stop words et lorsqu'il s'agit de la langue anglaise on n'aura pas à gérer les liaisons.

iii. Evaluation

Pour répondre au besoin de scalabilité, nous avons fait en sorte que notre système prenne en compte plusieurs langues (français et anglais pour le moment) et avons automatisé le processus de prétraitement par identification automatique de la langue de l'input.

Pour répondre au besoin de robustesse, nous avons testé notre algorithme sur des phrases de différentes longueurs (25 -> 500 caractères) et les résultats ont été satisfaisants.

Quoique nous avons essayé de contourner la contrainte de la variations des mots clés en utilisant le stemming, cette technique présente des limites. Par exemple les mots "serving" et "service" n'ont pas le même stem : 'serving' devient 'servic' alors que 'service' devient 'serv'. Par conséquent, dans le cas des outliers de ce type, nous avons une difficulté à déterminer la présence du mot clé.

c. Système de classification de l'aspect d'excuse

i. Besoin et spécifications

On se propose maintenant d'identifier la présence d'un aspect d'excuse dans une réponse donnée.

Ce système devra répondre à des contraintes de robustesse et de scalabilité. Ainsi que de pouvoir identifier cet aspect dans des phrases complexes et diverses structures grammaticales.

ii. Choix et stack technique

Notre premier réflexe a été de concevoir un modèle en utilisant l'architecture LSTM puisque les LSTM sont conçus pour **traiter les dépendances à long terme** dans les séquences de données, ce qui est crucial dans la compréhension du contexte dans lequel une excuse est exprimée.

En outre, les LSTM sont capables de gérer des **séquences de longueur variable**, ce qui peut être un avantage dans le traitement de phrases de différentes tailles. Cela peut être particulièrement utile dans le contexte de la classification d'excuses, où la longueur des phrases peut varier considérablement.

En plus, les LSTM sont bien adaptés **pour capturer les relations complexes entre les mots** dans une séquence. Cela permet au modèle de comprendre les nuances et les subtilités du langage, ce qui est important lors de la détection de la présence d'excuses qui peut souvent être **exprimée de manière contextuelle**.

Enfin, pour être honnête, les LSTM ont été pour nous **un choix sûr** puisque ils ont montré des performances solides dans la classification de séquences selon les recherches qui ont abouti à ce choix, nous avons jugé que leur utilisation répandue dans ce domaine peut être un indicateur de leur efficacité.

Cependant, lors du prétraitement des données dans le processus de conception de cet LSTM nous avons constaté que les mots utilisés pour exprimer l'excuse n'était pas vraiment aussi varié que ça ! on a constaté que les mots les plus utilisés, dans notre dataset pour les entrées classifiées positivement pour la présence d'aspect d'excuse, étaient : "Sorry", "Apology", "regret", "forgive", "excuse"

D'où vient l'idée de tester une solution complémentaire au modèle basé sur la recherche de ces mots là dans la réponse. Cette méthode là est inspirée par la méthode qu'on a utilisée dans la détection de mots clés.

iii. Méthode 1 : Rule based

1. Modélisation

Pour ce faire, nous avons implémenté la technique de stemming pour retrouver les racines de chaque mot de la réponse puis une simple recherche des mots clés que nous avons identifiés dans la réponse prétraité suffit pour identifier l'aspect d'excuse.

2. Evaluation

Pour évaluer cette méthode nous avons lancé un teste sur la totalité de la dataset étiqueté que nous allons utilisé pour entraîner notre LSTM et nous avons eu un taux de précision de 100%.

Malheureusement, par faute de temps, nous n'avons pas adapté cette technique à d'autres langues.

iv. Méthode 2 : Classifier binaire LSTM

1. Préparation des données

Faute de la présence d'une base de données de références, nous avons dû créer la nôtre en utilisant chatgpt. Nous avons créé 90 entrées avec le sorry aspect préalablement classifié.

Voici quelques lignes de ce dataset :

```
{
  "response": "Thank you for your feedback! We're delighted to hear that you enjoyed your experience at our restaurant. Your positive comments make our day! #HappyCustomer #ThankYou #CustomerSatisfaction",
  "sorry_aspect": 0
},
{
  "response": "We're sorry to hear that your experience fell short of expectations. Please let us know more details so we can address and improve upon the issues you encountered. Your satisfaction is our priority. #CustomerFeedback #ImprovementOpportunity",
  "sorry_aspect": 1
},
{
  "response": "We apologize for any inconvenience you may have experienced during your visit. Your feedback is crucial to us, and we're committed to making it right. Please DM us with more details so we can better understand and address your concerns. #CustomerService #Apology",
  "sorry_aspect": 1
}
```

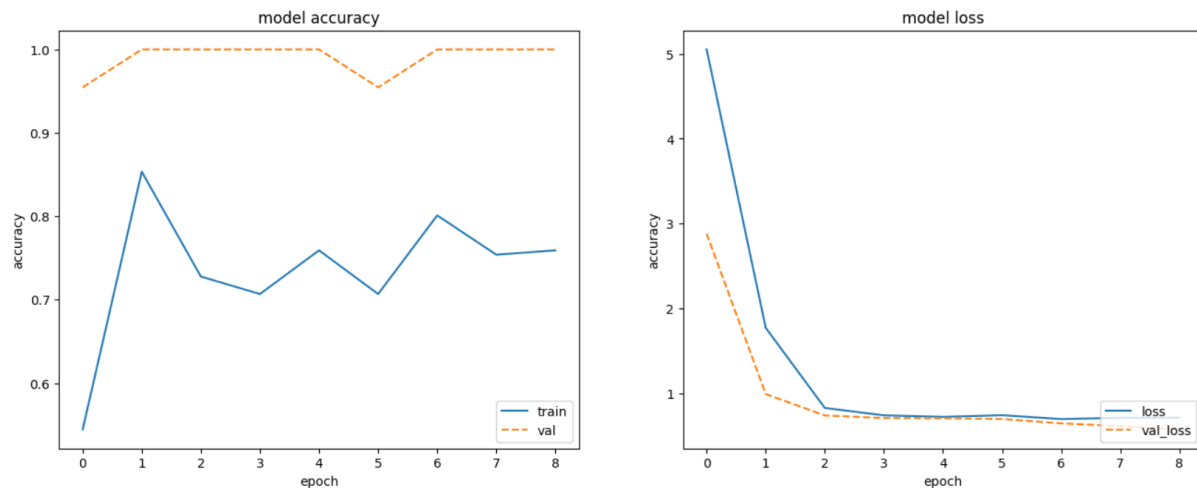
```

    },
    {
      "response": "Thank you for taking the time to share your thoughts. We appreciate both positive and constructive feedback. If there's anything specific you'd like us to look into or improve, please let us know. Your satisfaction is important to us! #FeedbackMatters",
      "sorry_aspect": 0
    },
  ],

```

Nous avons utilisé des techniques de data augmentation pour accroître le nombre d'entrées de notre base de données. Pour ce faire, nous avons créé des redondances aléatoires dans la base en gardant les classes équilibrées. La base résultante compte 213 entrées.

2. Evaluation



Vu que la base de données est assez petite nous avons évalué le modèle sur 10% ce qui est égale à une vingtaine d'entrées, c'est pour cela que le modèle a l'air d'être en surapprentissage.

Faute de temps, nous n'avons pas trouvé la chance de l'améliorer mieux que cela. Mais, pour le moment, avec une précision de 70%, ce modèle constitue un bon début. Voici quelques exemple de classifications que nous avons essayé :

```
print(predict("We apologize for this incident, we promise you a better experience on your next visit."))
print(predict("Your positive feedback motivates us to keep delivering exceptional service!"))
print(predict("We're sorry for any inconvenience during your recent interaction with our service."))
print(predict("Thank you for your positive review!"))
print(predict("Your feedback is essential to us! We're sorry if your recent experience didn't meet your expectations."))
print(predict("We're grateful for your kind words and look forward to creating more memorable experiences with you."))
```

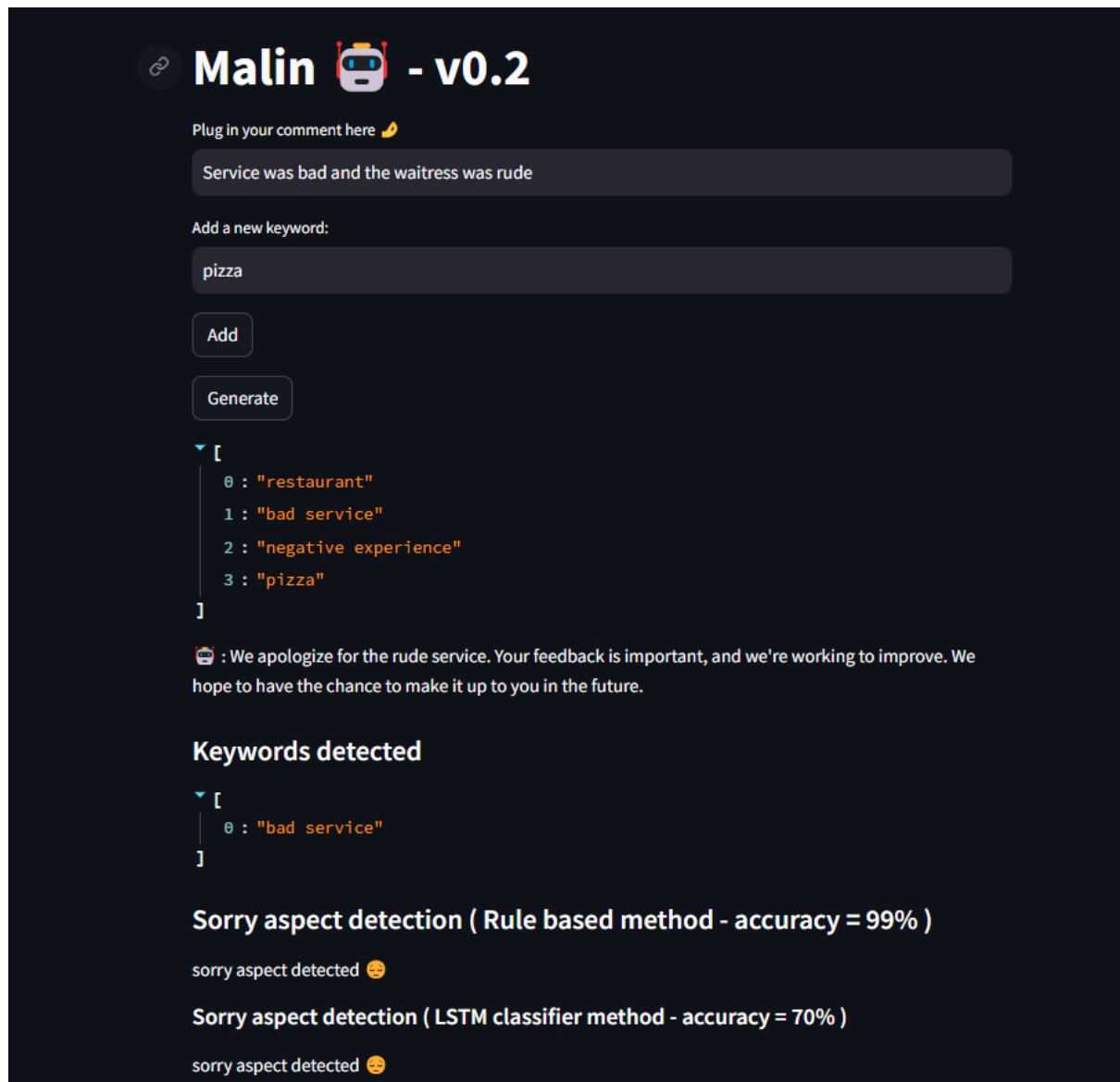
```
1/1 [=====] - 0s 27ms/step
apology
1/1 [=====] - 0s 26ms/step
no apology
1/1 [=====] - 0s 25ms/step
apology
1/1 [=====] - 0s 29ms/step
no apology
1/1 [=====] - 0s 37ms/step
apology
1/1 [=====] - 0s 25ms/step
no apology
```

3. Déploiement

Pour le déploiement de ce modèle, nous l'avons sauvegardé et intégré dans un environnement docker sur l'image officielle de tensorflow pour pouvoir utiliser les fonctions de la bibliothèque keras pour faire des prédictions.

Interface utilisateur de l'application

Pour englober les trois modules de notre application : générateur de réponses, détecteur de mots clés et classifieur de l'aspect d'excuse, nous utilisons streamlit : Un framework python qui permet de créer des applications web.



Déploiement du module IA

On utilise gitlab pour la planification de la pipeline CICD.

On définit trois états : Test, build et déploiement définis comme suite dans le fichier gitlab-ci.yaml

```


stages:
  - test
  - build
  - deploy

run_tests:
  stage: test
  image: tensorflow/tensorflow:latest
  script:
    - pip install -r requirements.txt
    - pytest test.py
build_image:
  stage: build
  image: docker:20.10.16
  services:
    - docker:20.10.16-dind
  variables:
    DOCKER_TLS_CERTDIR: "/certs"
  script:
    - export TMPDIR=$HOME/tmp
    - docker build -t kaiken26/projet-devops:mlops-pipeline-for-llm-testing-2.0 .
    - docker login -u $DOCKER_USER -p $DOCKER_PASS
    - docker push kaiken26/projet-devops:mlops-pipeline-for-llm-testing-2.0




deploy_kub:
  stage: deploy
  image:
    name: bitnami/kubectl:latest
    entrypoint: ['']
  script:
    - export KUBECONFIG=kubeconfig.txt
    - kubectl apply -f deployment.yml
    - kubectl apply -f service.yml
    - kubectl set image deployment/mlops-pipeline-for-llm-testing-deployment
mlops-pipeline-for-llm-testing=kaiken26/projet-devops:mlops-pipeline-for-llm-testing-2.0
    - kubectl events


```


Voici l'état de la pipeline :

 Passed

00:12:26
 1 day ago

Update file `.gitlab-ci.yml`
 #1119635729  main  961d2347 





On choisit pytest comme framework de test automatique qui sert en une derniers vérification en utilisant l'environnement ou en prévoit déployer l'application avant de créer l'image finale.

```
def test_process_input():

    # Test negative sentiment
    keywords = ["restaurant", "negative experience"]
    comment = "This place is way too overpriced for mediocre food."
    result = process_input(comment, keywords)
    assert result != ""

    # Test positive sentiment
    keywords = ["restaurant", "good experience"]
    comment = "If you want healthy authentic or ethic food, try this place."
    result = process_input(comment, keywords)
    assert result != ""

    # Test with empty input
    keywords = []
    comment = ""
    result = process_input(comment, keywords)
    assert result == "Please enter a comment and keywords"
```

Après la phase de test nous sommes en mesure de créer une image docker que nous allons conserver dans notre Docker Hub pour pouvoir la consommer ultérieurement en phase de déploiement.



kaiken26/projet-devops:mlops-pipeline-for-llm-testing-2.0

DIGEST: sha256:1e9f54dd708eefd6fcc21202704329083fd0b1611005c45d54b12ae4ed0d951a

OS/ARCH
linux/amd64

COMPRESSED SIZE ⓘ
2.88 GB

LAST PUSHED
12 hours ago by [kaiken26](#)

TYPE
Image

Nous choisissons le service azure kubernetes AKS pour le déploiement fluide de notre conteneur.

Pour ce faire, nous configurons nos manifest file deployment.yaml et service.yaml pour le déploiement de notre application comme suite :


```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: mlops-pipeline-for-llm-testing-deployment
  labels:
    app: mlops-pipeline-for-llm-testing
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mlops-pipeline-for-llm-testing
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
  template:
    metadata:
      labels:
        app: mlops-pipeline-for-llm-testing
    spec:
      containers:
        - name: mlops-pipeline-for-llm-testing
          image: kaiken26/projet-devops:mlops-pipeline-for-llm-testing-1.0
          ports:
            - containerPort: 8501

```

La partie en sélectionné en jaune montre la strategie de mise a jour que nous avons adopté. Ici le RollingUpdate.

Nous pouvons voir l'état de notre service avec la command `kubectl get svc` :

```

PS /home/aymen_fourati> kubectl get pods

```

NAME	READY	STATUS	RESTARTS	AGE
grafana-7c5df4775-nzrkc	1/1	Running	0	22h
mlops-pipeline-for-llm-testing-deployment-ffbd8ddcb-p6mc8	1/1	Running	0	22h
prometheus-alertmanager-0	1/1	Running	0	22h
prometheus-kube-state-metrics-85596bfdb6-z4jdx	1/1	Running	0	22h
prometheus-prometheus-node-exporter-7q6fc	1/1	Running	0	4m19s
prometheus-prometheus-node-exporter-qvq6g	1/1	Running	0	3m47s
prometheus-prometheus-pushgateway-79745d4495-mw4hb	1/1	Running	0	22h
prometheus-server-fd677cd4c-wq5d2	1/2	Running	0	22h