**جامعة تشرين كلية الهندسة المعلوماتية قسم الذكاء الصنعي**

الجمهورية العربية السورية

وزارة التعليم العالي

جامعة تشرين \_ كلية الهندسة المعلوماتية

قسم الذكاء الصنعي

**الترجمة من اللغة الإنكليزية إلى اللغة الفرنسية**

**Translation from English to French**

( مشروع فصلي )

**إعداد الطلاب :**

عمار معلا زين العابدين سلمان روان دريباتي

**إشراف :**

العام الدراسي 2023-2024

**ملخص**

في هذا المشروع، تم استخدام تقنيات التعلم الآلي لتحسين أداء الترجمة من اللغة الإنجليزية إلى اللغة الفرنسية، وتم مقارنة أداء شبكات النموذج العصبي العميق التكراري (RNN) مع وبدون تضمين الكلمات في المدخلات (embedding).

أظهرت النتائج أن استخدام تضمين الكلمات في المدخلات (embedding) يحسن أداء النماذج في الترجمة بشكل كبير، حيث تمكنت النماذج التي استخدمت التضمينات من تحقيق معدل دقة أفضل بكثير مقارنة بالنماذج التي لم تستخدم التضمينات.

**Abstract**

In this project, machine learning techniques were used to improve the performance of English to French translation, and the performance of Recurrent Neural Networks (RNNs) with and without word embedding in the input was compared.

The results showed that using word embedding in the input significantly improves the performance of the models in translation. Models that used embeddings were able to achieve much higher accuracy rates compared to models that did not use embeddings.

**الفهرس**

**1-الفصل الأول :** مقدمة..............................................................................1

1-1-الدراسات المرجعية..........................................................................1

1-1-1-الأوراق التي استندنا عليها خلال المشروع:............................................1

1-1-2- جدول التلخيص : ..................................................................4

2-1- البيئة والأدوات المستخدمة :................................................................5

3-1- منهجية البحث :...........................................................................6

**2-الفصل الثاني :** الجزء النظري :....................................................................7

2-1-خوارزميات الترجمة ما قبل 2014 :..........................................................8

2-1-1- Knowledge-based algorithms:.................................................9

2-1-2- Statistical Machine Translation :............................................10

2-2- خوارزميات الترجمة ما بعد 2014 : ........................................................11

2-2-1- Neural-based : .................................................................................11

2-2-2- Deep Learning-based Neural Machine Translation:.......................12

2-2-3- Artificial Intelligence-based Models :.......................................13

**3-الفصل الثالث :** الجزء العملي :..................................................................14

3-1-

3-1-1-

3-1-2-

3-2-النتائج:..................................................................................25

3-3-الصعوبات:..............................................................................29

3-4-التوصيات:...............................................................................30

**المراجع Refrences ..............................................................................**31

**فهرس الأشكال :**

الشكل 3-1 .......................................................................................12

الشكل 3-2 .......................................................................................13

الشكل 3-3 .......................................................................................14

الشكل 3-4 .......................................................................................16

الشكل 3-5 .......................................................................................16

الشكل 3-6 .......................................................................................17

الشكل 3-7 .......................................................................................17

الشكل 3-8 .......................................................................................13

الشكل 3-9 .......................................................................................20

الشكل 3-10 .....................................................................................24

**فهرس الجداول :**

الجدول(1-1) ........................................................................................5

الجدول(2-1) ......................................................................................12

**قائمة المصطلحات:**

|  |  |
| --- | --- |
| **المختصر الموافق** | **المصطلح** |
| RNN | Recurrent Neural Networks |
| DNN | Deep Neural Networks |
| LSTM | Long Short-Term Memory |
| MLOps | Machine Learning Operations |
| API | **Application Programming Interface** |
|  |  |
|  |  |

1. **الفصل الأول: مقدمة**

الترجمة من لغة إلى أخرى تعتبر واحدة من أهم مهام وتحديات تعلم الآلة ومعالجة اللغات الطبيعية. فاللغات الطبيعية تتضمن تنوعًا كبيرًا في النحو والمفردات والتعابير اللغوية، وهذا يجعل من الترجمة بين اللغات المختلفة مهمة صعبة.

تعتمد تقنيات الترجمة الحديثة على النماذج العميقة للتعلم الآلي، والتي تتضمن شبكات النموذج العصبي العميق (Deep Neural Networks) وخوارزميات تعلم الآلة المعتمدة على البيانات. وتستخدم هذه التقنيات مجموعات كبيرة من البيانات المترجمة لتدريب نماذج الترجمة الآلية، حيث يتم تدريب النموذج على تحويل النص المكتوب في اللغة الأولى (الإنجليزية) إلى النص المكتوب في اللغة الثانية (الفرنسية).

تعتمد نماذج الترجمة الحديثة على تقنيات مثل تضمين الكلمات (Word Embedding) لتحقيق أداء ممتاز في الترجمة الآلية. ومع تزايد حجم مجموعات البيانات المترجمة وتحسن تقنيات التعلم الآلي، يمكن توفير نتائج الترجمة الآلية التي تقارب جودة الترجمة البشرية في بعض الحالات، وهذا يجعل الترجمة الآلية أداة قيمة في مجالات الأعمال والتعليم والتواصل الدولي.

* 1. **الدراسات المرجعية :**

**1-1-1-الأوراق التي استندنا عليها خلال المشروع:**

Efficient Estimation of Word Representations in Vector Space-

في هذه الورقة تم انتاج طريقة عصرية لتمثيل النص ضمن vector space structured هيكليته تعتمد

على العلاقات الدلالية بين الكلمات أو بشكل أكثر وضوح كل كلمة سيتم ربطها بشعاع ثم إسقاطها في هذا

الفضاء بحيث تكون العلاقات الهندسية بين word vectors انعكاس للعلاقات الدلالية بين الكلمات, وتعتبر

هذه التقنية أساسية في تطوير نماذج الترجمة الألية باستخدام الشبكات العصبونية, حيث يتم استخدام

word embedding لتمثيل الجمل في شكل رقمي بحيث يمكن للشبكات العصبية فهم المعنى والمضمون الخاص بالجمل والكلمات.

تم تأليف الورقة من قبل Thomas Mikolov, Quoc Le, Edward H ، وتم نشرها في عام 2013 في مؤتمر تقنيات اللغة الطبيعية.

- Sequence to Sequence Learning with Neural Networks

في هذه الورقة تم استخدام الشبكات العصبونية LSTM للترجمة من جملة إلى أخرى معتمدين على نموذج

Sequence-to-Sequence حيث يتم في البداية استخدام word embedding لتمثيل الجمل ثم يتم

إدخال هذه المتجهات إلى شبكة عصبونية والتي تقوم بتحويل هذه المتجهات إلى المتجهات الخاصة بالجملة

الهدف وبعد ذلك يتم تحويل المتجهات الناتجة إلى كلمات حقيقية في اللغة المستهدفة وتم الاعتماد أيضا

على طبقة Attention للتركيز على الكلمات الأكثر أهمية من خلال تخصيص وزن معين لكل كلمة في الجملة.

تم تأليف الورقة من قبل Ilya Sutskever, Oriol Vinyals,Quoc V. Le ، وتم نشرها في عام 2014.

- Neural Machine Translation by Jointly Learning to Align and Translate

في هذه الورقة تم استخدام نفس الآلية التي تم الاعتماد عليها في الورقة السابقة من استخدام تقنية التضمين و شبكة LSTM وكذلك استخدام طبقة Attention للتركيز على الكلمات الأكثر أهمية ومنعا لحدوث أخطاء في الترجمة مثل وضع كلمة ترجمه بغير موضعها الصحيح مثل لو كانت جملة الترجمة "الكتاب على الطاولة" فتكون الترجمة "The table is on the book" بدلا من "The book is on the table"

وهذا خاطئ لأنه تم وضع كلمتين بغير موضعها الصحيح لذلك تم استخدام طبقة Alignment لتحديد الموضع المناسب لكل كلمة اعتماد على درجات التركيز التي تأتي طبقة Attention.

تم تأليف الورقة من قبل Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio , وتم نشرها في عام 2014.

|  |  |  |  |
| --- | --- | --- | --- |
| **مساؤى** | **محاسن** | **الأفكار الرئيسية** | **اسم الورقة البحثية** |
| قد تعاني طريقة التضمين من بعض المشاكل في تمثيل الكلمات النادرة ويحتاج أيضا مصادر بيانات كبيرة | تقليل حجم النماذج اللازمة لتمثيل الكلمات وتحسين وقت التدريب للنماذج العصبية , مما يساعد في تحسين أداء تطبيقات الترجمة الآلية | استخدام تقنية التضمين لتحسين جودة الترجمة الآلية باستخدام الشبكات العصبونية | Efficient Estimation of Word Representations in Vector Space |
| تعقيد عملية التدريب والحاجة إلى كميات كبيرة من البيانات لضمان تحقيق النتائج المرجوة وظهور الأخطاء في الترجمة الآلية والتي تؤثر على جودة الترجمة وتفاصيل المعنى | القدرة على التعامل مع مشكلة طول الجمل والتعامل مع ترجمة الجمل ذات المعاني المجردة والمعقدة بفضل طبقة attention | استخدام الشبكة العصبونية LSTM مع تقنية التضمين وتقنية attention للتركيز على الكلمات الأكثر أهمية بالنسبة للجملة المستهدفة | Sequence to Sequence Learning with Neural Networks |
| على الرغم من استخدام Alignmentإلا أن النموذج يعاني من مشكلة في التعامل مع الجمل المعقدة والطويلة وخاصة الجمل التي تحوي كلمات متعددة المعاني حيث تؤدي التداخلات بين الكلمات إلى تحديد مواضع غير ملائمة للكلمات | تحسين جودة ودقة الترجمة والتغلب على مشكلة الترجمة الحرفية بعد استخدام تقنية Alignment | استخدام الشبكة العصبونية LSTM مع تقنية التضمين وتقنية attention وتقنية Alignment لتحديد الموضع المناسب للكلمات في الجملة الهدف بشكل متزامن | Neural Machine Translation by Jointly Learning to Align and Translate |

**-2-1البيئة والأدوات المستخدمة**

في هذه الفقرة، سنشرح بإيجاز عن الأدوات والبيئة التي تم استخدامها في هذا البحث، والتي تشمل

Github Actions، Docker، Python، Jupyter Notebook، FastAPI، Colab، TensorFlow، Keras، NumPy، PyYAML، Pickle.

* Github Actions هو خدمة تقدمها Github لإدارة عمليات التطوير الخاصة بمشاريع البرمجة. باستخدام Github Actions يمكنك إنشاء نظام للتحكم بعمليات التطوير والإختبار والنشر بصورة آلية، بحيث يمكنك تحسين الإنتاجية وتوفير الوقت اللازم لإنجاز المهام. يتم إعداد Github Actions بإستخدام ملفات YAML التي تحتوي على سلسلة من الخطوات التي يجب تنفيذها.
* Docker هو برنامج يساعد على إنشاء وتشغيل بيئات البرمجة في أي نظام تشغيل، بما في ذلك Linux وWindows وMacOS. يساعد Docker في توفير بيئة متكاملة للتطوير والتشغيل، ويسمح بإنشاء حاويات (containers) تحتوي على جميع المكتبات والأدوات والإعدادات اللازمة لتشغيل تطبيقك.
* Python هي لغة برمجة شائعة الاستخدام ومفتوحة المصدر، وتستخدم في العديد من المجالات بما في ذلك الذكاء الاصطناعي وعلوم البيانات وتطوير الويب.
* Jupyter Notebook هو برنامج تفاعلي للبرمجة يسمح للمستخدمين بكتابة الشفرات وعرض النتائج في واجهة مستخدم بديهية. يتم استخدام Jupyter Notebook عادة للتعلم العملي والتجارب السريعة في علوم البيانات والذكاء الاصطناعي.
* Colab هو بيئة تفاعلية تقدمها Google، ويتم استخدامها عادة لتطوير النماذج الذكية والتعلم العميق وعلوم البيانات. يتميز Colab بواجهة مستخدم بديهية وخدمات السحابة التي يتم توفيرها من قبل Google.
* TensorFlow هو إطار عمل للتعلم الآلي والذكاء الاصطناعي، ويستخدم عادة لبناء الشبكات العصبية وتدريبها.
* Keras هو إطار عمل للتعلم الآلي يُستخدم كواجهة عالية المستوى لتطبيقات TensorFlow. يعتبر Keras سهل الاستخدام ويتميز بالمرونة والسرعة في بناء الشبكات العصبية وتدريبها.
* NumPy هو مكتبة لغة بايثون مفتوحة المصدر تُستخدم للعمليات الرياضية والعلمية، وتتضمن مجموعة من الأدوات والوظائف المفيدة لتحليل ومعالجة البيانات الكبيرة. يتميز NumPy بالسرعة والكفاءة والتعامل مع البيانات في صيغة المصفوفات.
* yaml هو مكتبة لغة بايثون مفتوحة المصدر يتم استخدامها للتعامل مع صيغة YAML يستخدم YAML عادة لتخزين البيانات المنظمة بشكل هرمي، ويمكن استخدام yaml لتحويل البيانات بين صيغ YAML وصيغ أخرى.
* Pickle هو مكتبة لغة بايثون تُستخدم للتسلسل والإلغاء التسلسل للبيانات. يستخدم Pickle عادة لحفظ واستعادة البيانات في صيغة ملف، ويتميز بالسهولة والكفاءة في التعامل مع البيانات.
* Neptune.ai هو إطار عمل يستخدم لتتبع ومراقبة نماذج الذكاء الصنعي في مرحلة التعلم.
* FastAPI هو إطار عمل يستخدم لبناء Restful Api لتحويل النماذج إلى micro service.

**-3-1 منهجية البحث**

Dataset: English to Frence

معالجة الداتا

استخدام الشبكة العصبونية

LSTM or RNN

تحويل الجملة الانكليزية إلى الفرنسية

Dataset: English to Frence

معالجة الداتا

استخدام تقنية التضمين   
Word Embedding

استخدام الشبكة العصبونية

LSTM or RNN

تحويل الجملة الانكليزية إلى الفرنسية

**2-الفصل الثاني : الجزء النظري :**

بداية سنتحدث عن الخوارزميات التي استخدمت في مشروع الترجمة من اللغة الإنكليزية إلى اللغة الفرنسية من الأقدم للأحدث , حيث قام العلماء بتطوير الخوارزميات انطلاقآ من المشاكل الظاهرة محاولة تخفيفها.

يوجد فترتان تاريخيتان للترجمة من اللغة الإنكليزية إلى اللغة الفرنسية , قبل 2014 وفترة الترجمة القائمة على التعلم العميق بعد 2014 .

في أي ترجمة، سواء كانت بشرية أو آلية، يجب أن ينقل معنى النص في لغة المصدر (الأصلية) بالكامل إلى معناه المعادل في ترجمة اللغة الهدف. وبينما يبدو هذا واضحا ظاهريا، فهو غالبا ما يكون أكثر تعقيدا. فالترجمة ليست مجرد استبدال كلمة بكلمة.

يجب على المترجم البشري تفسير وتحليل جميع العناصر داخل النص وفهم كيف يمكن لكل كلمة أن تؤثر على سياق النص. وهذا يتطلب خبرة واسعة في قواعد اللغة، وبناء الجمل (تراكيب الجمل)، والدلالات (المعاني)، وما إلى ذلك، في اللغتين، المصدر والهدف، بالاضافة إلى الخبرة في مجال النص.

كما أن لكل من الترجمة البشرية والآلية حصتهما من التحديات. على سبيل المثال، لن ينتج اثنين من المترجمين ترجمات متطابقة للنص نفسه في نفس اللغتين، كما أن الأمر قد يستغرق جولات تنقيح عديدة لتلبية متطلبات العميل. أما الترجمات الآلية فتواجهها صعوبات في ترجمة العناصر السياقية والثقافية للنص، وتعتمد جودتها على نوع النظام وكيفية تدريبه، إلا أنها فعالة للغاية بالنسبة لأنواع معينة من المحتوى وحالات الاستخدام، التي تشمل على سبيل المثال، النصوص المتكررة، واللغة المنظمة، وغيرها الكثير.

قبل عام 2014 كانت الخوارزميات الأساسية المستخدمة في مشاريع الترجمة الآلية من اللغة الانكليزية إلى اللغة الفرنسية هي :

**2-1-خوارزميات الترجمة ما قبل 2014 :**

**2-1-1- Knowledge-based algorithms**: تستخدم هذه الخوارزمية مجموعة من القواعد اللغوية والقواعد النحوية لتحليل النص الأصلي وتوليد الترجمة المناسبة. ومن الأمثلة على هذه الخوارزميات الشهيرة هي SYSTRAN وLOGOS .

**2-1-2- Statistical Machine Translation :** تستخدم الترجمة الآلية الإحصائية نماذج الترجمة الإحصائية الناتجة عن تحليل بيانات التدريب أحادية وثنائية اللغة. ويستخدم هذا النهج أساسا القدرة الحاسوبية لبناء نماذج بيانات معقدة لترجمة لغة مصدر إلى لغة أخرى. يتم اختيار الترجمة من بيانات التدريب باستخدام خوارزميات لتحديد الكلمات أو العبارات الأكثر تواترا , يعتبر بناء نماذج الترجمة الآلية الاحصائية عملية سريعة وبسيطة نسبيا , تستخدم هذه التقنية وحدة المعالجة المركزية بشكل مكثف، وتتطلب تكوين أجهزة واسعة لتشغيل نماذج الترجمة بمستويات أداء مقبولة.

**2-2-خوارزميات الترجمة ما بعد 2014 :**

مننذ عام 2014 تم تطوير العديد من الخوارزميات الجديدة في مجال الترجمة الآلية من اللغة الانكليزية إلى اللغة الفرنسية , أصبحت تعتمد على Neural Network , Deep learning , من بين هذه الخوارزميات:

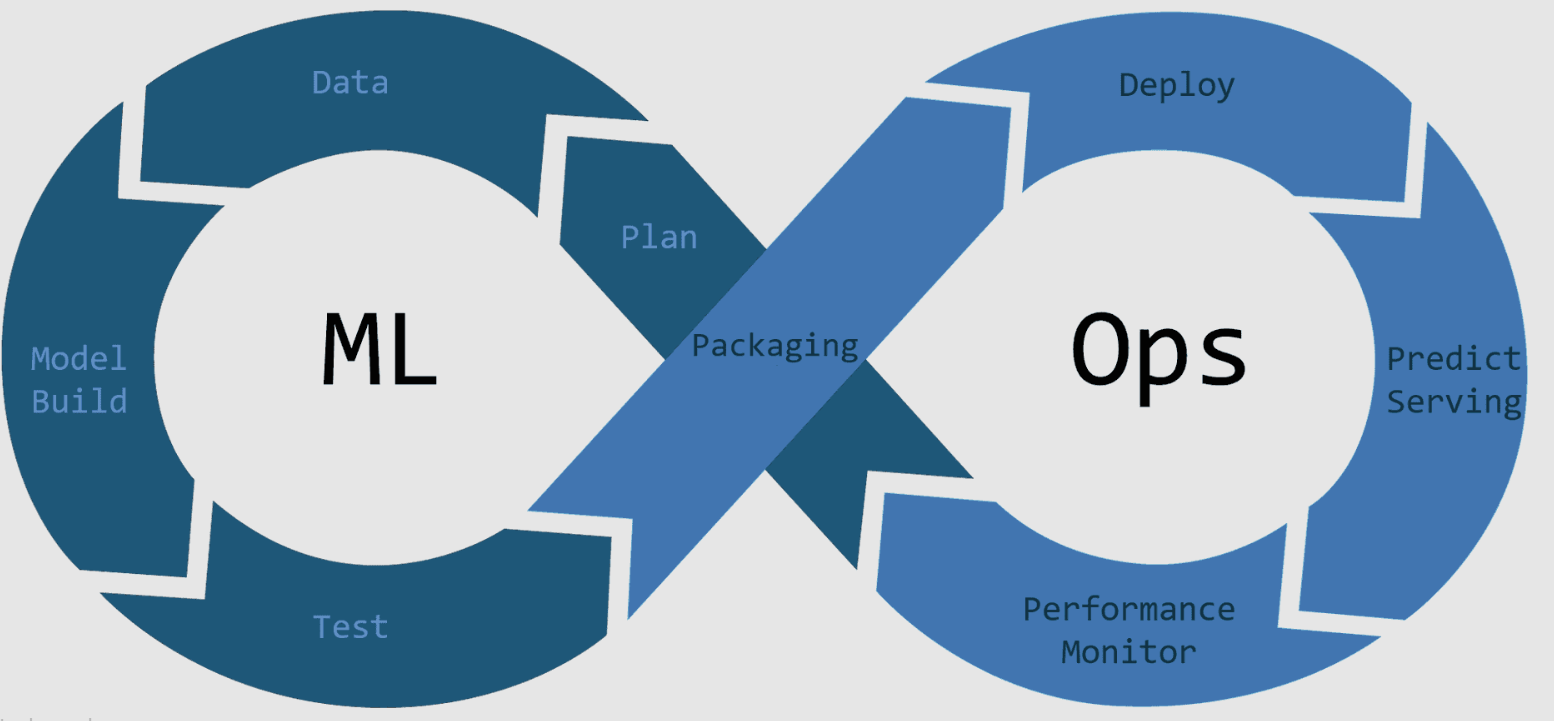
**2-2-1**- **Neural-based :** تستخدم هذه الخوارزمية شبكات عصبية لتحليل النص الأصلي وتوليد الترجمة المناسبة. وقد بدأ استخدام هذه الخوارزمية في مشاريع الترجمة الآلية منذ العام 2014، ومن الأمثلة الشهيرة على هذه الخوارزميات هي Google Neural Machine Translation وMicrosoft Translator .

**2-2-2- Deep Learning-based Neural Machine Translation :** وهي خوارزمية تستخدم شبكات عصبية عميقة لتحليل النص الأصلي وتوليد الترجمة المناسبة. ومن الأمثلة الشهيرة على هذه الخوارزميات هي BERT وGPT-3 .

**2-2-3- Artificial Intelligence-based Models :** وهي خوارزمية تستخدم تقنيات الذكاء الاصطناعي الأخرى بالإضافة إلى التعلم العميق لتحليل النص الأصلي وتوليد الترجمة المناسبة. ومن الأمثلة الشهيرة على هذه الخوارزميات هي IBM Watson Language Translator .

**3-الفصل الثالث: الجزء العملي**

تحدثنا في فقرة منهجية البحث عن عملية الترجمة و التي تبدأ بعملية المعالجة وتنتهي بمرحلة الحصول على model قادر على تحويل الجملة الانكليزية إلى فرنسية ولكن في هذا الفصل سوف نتحدث عن مفهوم أشمل وهو MLOps cycle و هي باختصار دورة حياة نموذج التعلم الآلي من مرحلة التخطيط plan إلى مرحلة الانتاج deployment والذي يوضحه الشكل التالي



الشكل(كذا)

**المرحلة الأولى: التخطيط planning**

في هذه المرحلة تم العمل على :

* تقسيم أعضاء الفريق إلى عدة فرق:

1. فريق البحث (Research Team) المسؤول عن وضع معمارية النماذج و التحكم بمعاملات التدريب وتدريب النماذج ومراقبة تدريب النماذج.
2. فريق (MLOps Team) المسؤول عن التعامل مع جميع الفرق وتأمين متطلباتهم من توفير أدوات لجعل مرحلة التدريب automation و مراقبة النماذج monitor والعمل على إيصال النماذج إلى مرحلة production.
3. فريق الاختبار (Testing Team) المسؤول عن اختبار النماذج والتأكد من كفاءتها.

* تقسيم بيئة العمل إلى بيئتين:

1. بيئة التطوير التدريب (Dev/Train)، وهنا سيتم تقسيم هذه المرحلة إلى بيئتين:

بيئة التدريب train وهي تتم على سيرفر التدريب الذي يحوي GPU وعلى اعتبار لا يوجد سيرفر مستقل لدينا, تم اعتبار سيرفر colab هو السيرفر الخاص بعملية التدريب.

بيئة التطوير dev وهي تتم على سيرفر مستقل (أجهزة الحاسب الخاص بنا) حيث يتم تحويل الأوزان الناتجة عن مرحلة التدريب إلى (Web Service).

1. بيئة الانتاج (Prod) حيث يتم فيها نشر (Web Service) على سيرفر production واختبارها وتحسينها.

* التركيز على معمارية المشروع وفقا لأفضل الممارسات في السوق الحالية, واخترنا التقسيم التالي تبعنا لاحتياجات المشروع:

1. ملف preprocess.py: يحوي على التوابع التي تقوم بمعالجة الداتا.
2. ملف test\_preprocess.py: يحوي على اختبار لكل تابع في ملف preprocess.py.
3. ملف models.py يحوي على عدة معماريات يمكن تحقيق أي معمارية للشبكة في هذا الملف.
4. ملف train.yaml: يحوي على إعدادات الخاصة بعملية التدريب (عقل المشروع) مثل نوع معمارية النموذج ومكان وجود الداتا الخاصة بالجمل الانكليزية والفرنسية ومعدل التعلم وعدد العينات وbatch size و عدد epochs ونسبة validation من الداتا ونوع الجهاز الذي سيتم التدريب عليه GPU or CPU.
5. ملف train.py: هو الملف الذي تتم فيه عملية التدريب بعد تحديد الإعدادات في train.yaml من معالجة الداتا إلى توفير أوزان خاصة بعملية المعالجة وأوزان خاصة بعملية التدريب.
6. ملف :inference.py يحتوى على اختبار بسيط للنموذج المدرب.
7. ملف :requirements.txt يحوى على python packages الخاصة بالمشروع مع رقم الأصدار لكل packages.
8. ملف Dockerfile: يحوي على الإعدادات الخاصة بإنشاء بيئة معزولة لتشغيل المشروع من نظام تشغيل وكل المتطلبات التي يحتاجها المشروع بحيث يصبح ك service قائمة بحد ذاتها.
9. ملف start.sh:
10. مجلد .github: يحوي على ملفات بصيغة yaml تستخدم في جعل عملية التدريب والتطوير والإنتاج بطريقة مؤتمتة automation.
11. مجلد :data يحوي على الداتا المستخدمة في المشروع والتي تضم الجمل الانكليزية مع مقابلاتها من اللغة الفرنسية.
12. مجلد fastapi: يحتوى على الملفات الخاصة بتحويل model إلى (Web Service).
13. مجلد :outputs يحتوي على الأوزان الخاصة بعملية المعالجة والتدريب
14. مجلد papers\_research: يحتوي على الأوراق البحثية التي تم اعتمادها في المشروع

إلى هنا نكون قد انتهينا من مرحلة التخطيط (planning) وسوف ننتقل إلى المرحلة الثانية الخاصة بتعريف الداتا ومعالجتها Data and Processing)).

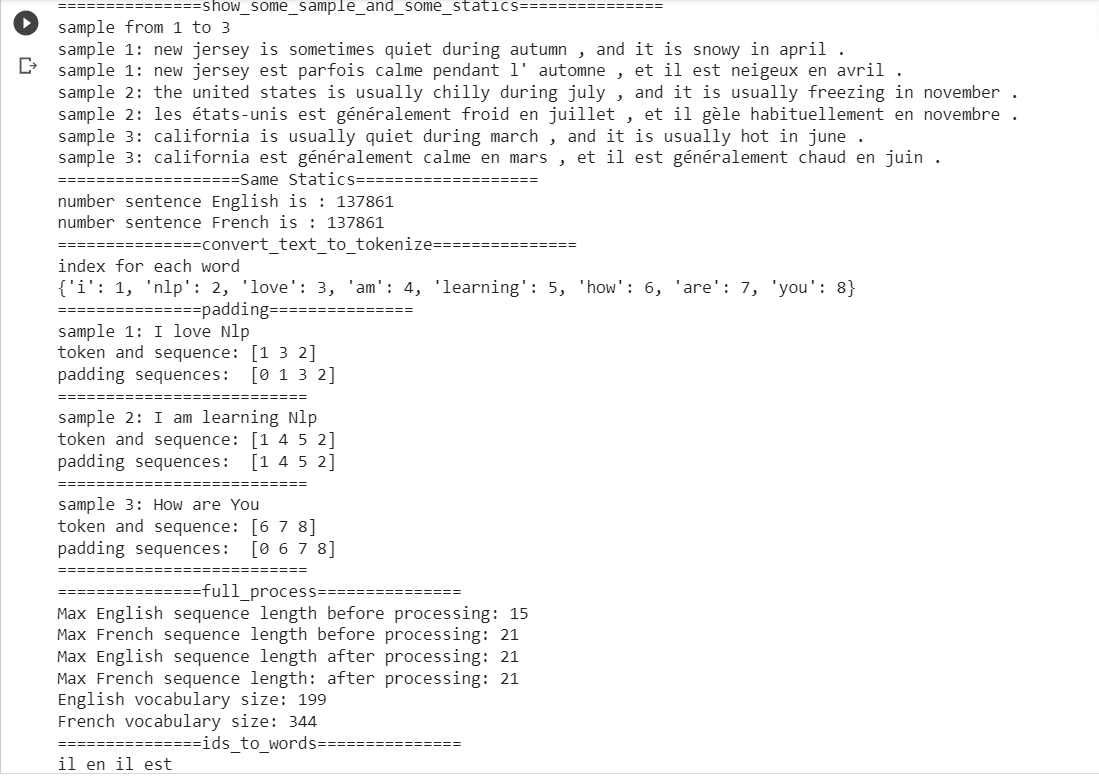
**المرحلة الثانية: الداتا ومعالجتها Data and Processing**

في هذه المرحلة تم العمل على :

* التعرف على الداتا المستخدمة في عملية التدريب والتي توجد في المجلد data حيث يوجد ملفين ملف خاص بالجمل الانكليزية small\_vocab\_en وملف خاص بالجمل الفرنسية small\_vocab\_fr وعدد الجمل في كل منهما هو137861**.**
* تعريف التابع max\_seq في ملف preprocess.py ومهمة هذا التابع هوتحديد أكبر طول جملة

قبل معالجة الداتا لكي تكون الجمل ذات طول واحد.

* تعريف التابع read\_data في ملف preprocess.py ومهمة هذا التابع هو قراءة كل سطر في الملف وتخزين الجمل في متحول نهائي ويأخذ المسار الكامل للملف كمتحول.
* تعريف التابع show\_some\_sample\_and\_some\_statics في ملف preprocess.py ومهمة هذا التابع بقراءة ملفsmall\_vocab\_en, small\_vocab\_frثم طباعة مجموعة من الجمل كل جملة انكليزية ومقابلتها من اللغة الفرنسيةوبعدها طباعة عدد الجمل الموجودة في كل ملف ويتم اختبار التابع في ملف test\_preprocess.py كما في الشكل (1)**.**
* تعريف التابع convert\_text\_to\_tokenize في ملف preprocess.py ومهمة هذا التابع هو بناء قاموس لمجموعة الجمل التي يتم إعطاءها للتابع وكذلك ترميز مجموعة الجمل بعد بناء هذا القاموس وتم اختبار التابع في ملف test\_preprocess.py كما في الشكل (1)**.**
* تعريف التابع padding في ملف preprocess.py ومهمة هذا التابع هو جعل سلسلة الجمل ذات طول واحد عبر إضافة أصفار أما على يمين السلسلة أو يسارها ويتم التحكم بالطول النهائي باستخدام المتحول length\_of\_pad وكذلك type\_pad لاختيار يمين أو يسار وتم اختبار التابع في ملف test\_preprocess.py كما في الشكل (1)**.**
* تعريف التابع full\_process في ملف preprocess.py ومهمة هذا التابع هو استخدام التوابع السابقة في معالجة الداتا بشكل كامل من تحديد قاموس لكامل الداتا (Tokenizer) ثم مرحلة (padding**)** لجعل الجمل بنفس الطول ومن ثم تعديل أبعاد أحجام المصفوفات الناتجة لكي تلائم شكل الدخل والخرج للشبكة العصبونية وتم اختبار التابع في ملف test\_preprocess.py كما في الشكل (1)**.**
* تعريف التابع ids\_to\_words في ملف preprocess.py ومهمة هذا التابع هي تحويل أرقام كل خرج الشبكة إلى ما يقابلها من كلمات في اللغة الفرنسية وتم اختبار التابع في ملف test\_preprocess.py كما في الشكل (1)**.**
* بالإضافة إلى اختبار التوابع السابقة في ملف test\_preprocess.py كما في الشكل (1) كذلك تم طباعة بعض العبارات التوضيحية لفهم الخرج بشكل أفضل.



الشكل(1)

إلى هنا نكون قد انتهينا من مرحلة الداتا ومعالجتها Data and Processing وسوف ننتقل إلى المرحلة الثالثة الخاصة ببناء عدة معماريات للنموذج وتدريبها (Build and Train Models).

**المرحلة الثالثة: بناء وتدريب النماذج Build and Train Models**

في هذه المرحلة تم العمل على :

* في ملف train.yaml كما تحدثنا في مرحلة التخطيط يحوي على إعدادات الخاصة بعملية التدريب (عقل المشروع) والذي يستطيع الباحث أو المطور التحكم بكامل عملية التدريب من خلاله حيث تم تعريف المتحولات التالية :

1. en\_url: هو متحول string يمثل مسار الملف الذي يحوي الجمل باللغة الانكليزية.
2. fr\_url: هو متحول string يمثل مسار الملف الذي يحوي الجمل المقابلة باللغة الفرنسية.
3. type\_model: هو متحول string يحوي على اسم معمارية الشبكة التي ستقوم عملية  
   التدريب عليها ونستطيع استخدام أحد هذه المعماريات عبر تغير الاسم فقط وتحوي أربع قيم :

Rnnتمثل شبكة RNN.

Rnn\_Embd تمثل شبكة RNN مع استخدام Word Embedding)).

Lstmتمثل شبكة LSTM.

Lstm\_Embd تمثل شبكة LSTM مع استخدام Word Embedding)).

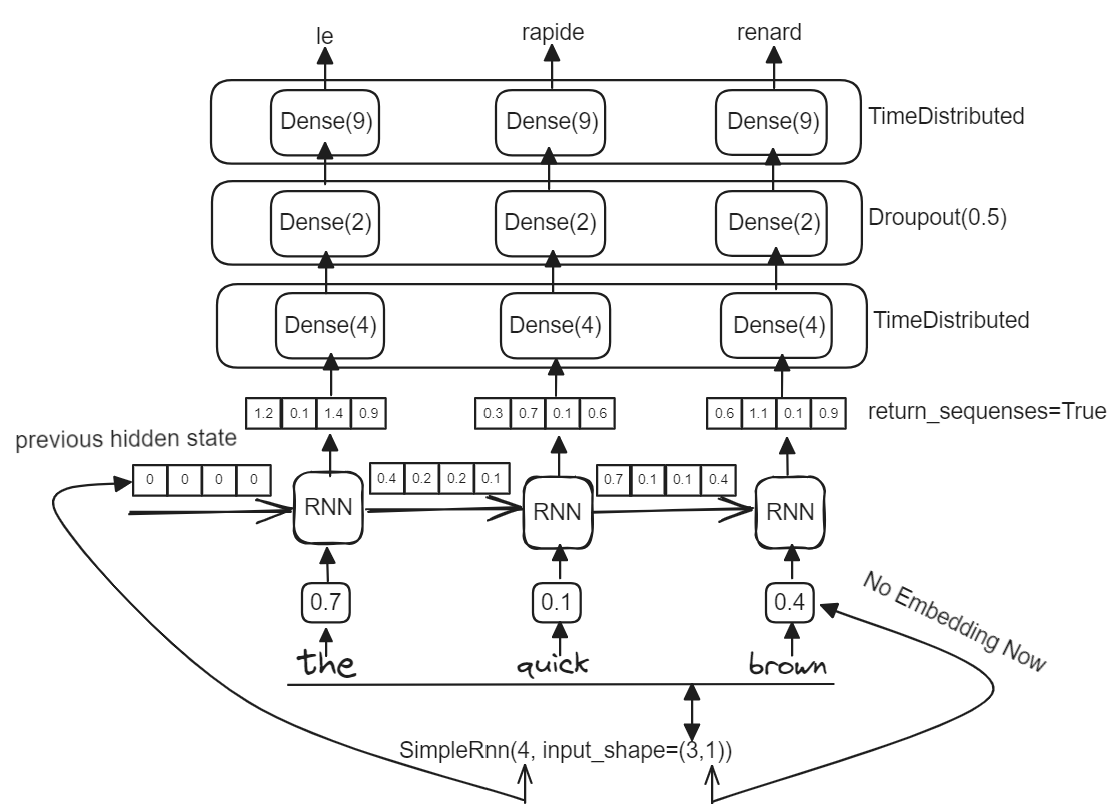
1. Learning\_rate : هو متحول float يمثل معدل التعلم يفضل أن تكون قيمته بين 0 و 1.
2. number\_sample: هو متحول integer يمثل عدد العينات المستخدمة في عملية التدريب.
3. batch\_size: هو متحول integer يمثل عدد العينات التي يتم إرسالها في كل دفعة خلال عملية التدريب يفضل أن يكون صغير لكي لتجنب استخدام الذاكرة بشكل كبير.
4. epochs: هو متحول integer يمثل عدد مرات تدريب النموذج على مجموعة العينات.
5. validation\_split: هو متحول float يمثل نسبة داتا الاختبار من كامل الداتا المستخدمة.
6. type\_device: هو متحول string لتحديد على ماذا تتم عملية التدريب سواء على CPU أو GPU وذلك بحسب التوافرية لدى الباحث أو المطور ويتم استخدام نفس الاسماء لتحديد نوع الجهاز.
7. length\_vector\_word: هو متحول integer يمثل طول شعاع الكلمة الذي نريد الحصول عليه في طبقة (Word Embedding).
8. n\_neuros\_rnn: هو متحول integer يمثل عدد الحالات المخفية state hidden التي تعطيها كل طبقة (RNN) للطبقة التالية .
9. n\_neuros\_lstm: هو متحول integer يمثل عدد الحالات المخفية state hidden التي تعطيها كل طبقة (LSTM) للطبقة التالية .
10. n\_neuros\_ timedistributed: هو متحول integer يمثل عدد العصبونات في طبقة (Time Distributed).

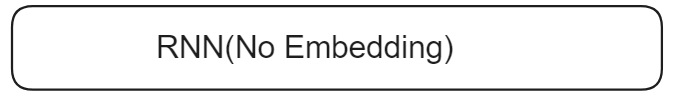
* في ملف models.py كما تحدثنا في مرحلة التخطيط أن هذا الملف سوف يستخدم لتحقيق أي معمارية للشبكة, لذلك تم العمل على:

1. بناء تابع motor يأخذ هذا التابع المتحولات الخاصة ببناء معمارية الشبكة مثل**:**
2. حجم مصفوفة الدخل الخاصة باللغة الانكليزية input\_shape
3. عدد كلمات اللغة الانجليزية الموجودة في الداتا المستخدمة dict\_en\_size
4. عدد كلمات اللغة الفرنسية الموجودة في الداتا المستخدمة dict\_fr\_size
5. باقي المتحولات تم شرحها في ملف train.yaml تحت بند Structural Models.
6. في التابع السابق تم تحقيق 4 معماريات للشبكة كل معمارية يتم أخذها باستخدام type\_model الموجود في ملف train.yaml, في حال كان يساوي:
7. type\_model="Rnn" اذا الشبكة المستخدمة هي RNN ولا يوجد طبقة  
   (Word Embedding), والطبقات على الترتيب التالي:
8. طبقة SimpleRnn حيث تأخذ n\_neuros\_rnn الذي يمثل عدد الحالات المخفية بين الطبقات و input\_shape ثنائية الجزء الأول يضم عدد الكلمات في الجملة ومثل أيضا time steps عدد العصبونات الموجود في طبقة rnn والتي سوف يهتم كل عصبون فيها بكلمة واحدة من الجملة والجزء الأخر من الثنائية يمثل طول شعاع الكلمة و تم وضع return\_sequences على القيمة true لكي يقوم كل عصبون بإرجاع hidden states الخاص به, الشكل (كذا) يوضح آلية الطبقة.
9. طبقة TimeDistributed تكمن فائدة هذه الطبقة في أنها لو لم تكن موجوده كان سيتم فقط معالجة hidden state لأخر عصبون وهذا سيئ لأننا نريد تحديد كلمة الخرج باللغة الفرنسية بناء على الكلمة الانكليزية الحالية التي وصلنا لها وليس بناء على كامل الجملة (بكلمات أخرى لو كانت المهمة تصنيف الجمل إلى خبر كاذب وحقيقي أي 0 أو 1 كان باستطاعتنا استخدام Dense على أخرhidden state خارج من الطبقة الأخيرة لأن المشكلة هي Many-To-One ولكن في حالتنا باستخدام TimeDistributed طبقنا Dense على كل hidden state لأن المشكلةMany-To-Many )

هذه الطبقة تأخذ n\_neurons\_ timedistributed كعدد عصبونات لكل dense, الشكل (كذا) يوضح آلية الطبقة.

1. طبقة Dropout مع قيمة 0.5لمنع حصول over fitting في الشبكة, الشكل (كذا) يوضح آلية الطبقة.
2. طبقة TimeDistributed ولكن نقوم بأعطاءه dict\_fr\_size (عدد كلمات اللغة الفرنسية في الداتا) كعدد عصبونات لكل dense مع تابع التفعيل softmax لإعطاء مصفوفة احتمالات للكلمات في كل dense وبعدها يتم اختيار الكلمة اعتماد على التابع ids\_to\_words المعرف مسبقا في مرحلة المعالجة والذي يقوم بتحويل كل مصفوفة احتمالات بعد اختيار أكبرها باستخدام argmax إلى الكلمة الفرنسية المقابلة للرقم اعتماد على القاموس , الشكل (كذا) يوضح آلية الطبقة.



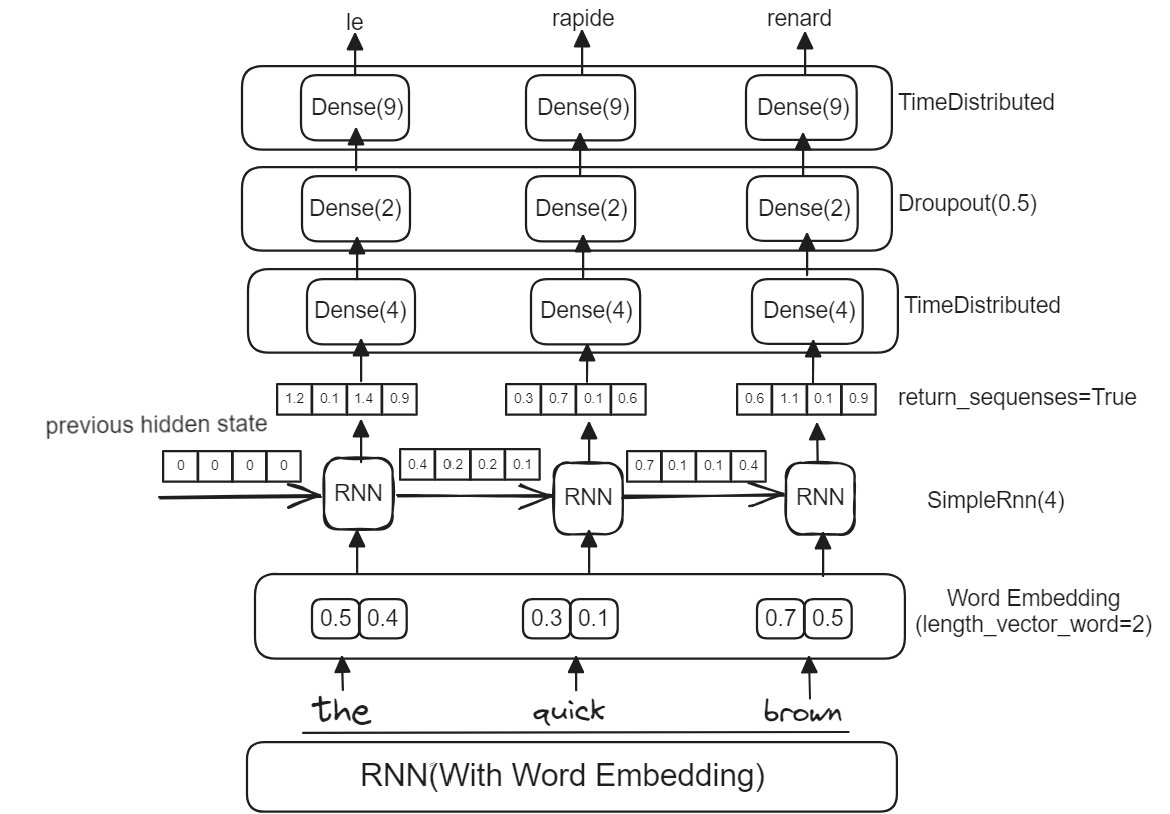


الشكل (كذا)

1. type\_model="Rnn\_Embd" اذا الشبكة المستخدمة هي RNN مع طبقة  
   (Word Embedding), والطبقات على الترتيب التالي:
2. طبقة Embedding حيث تأخذ dict\_en\_sizeالذي يمثل عدد الكلمات الانجليزية المستخدمة في الداتا و length\_vector\_word تم شرحه سابقا في ملف train.yaml يمثل طول شعاع الكلمة الذي نريد الحصول عليه و input\_length يمثل طول أكبر جملة موجود في الداتا سواء جملة فرنسية أو انكليزية وهي 21 في حالتنا

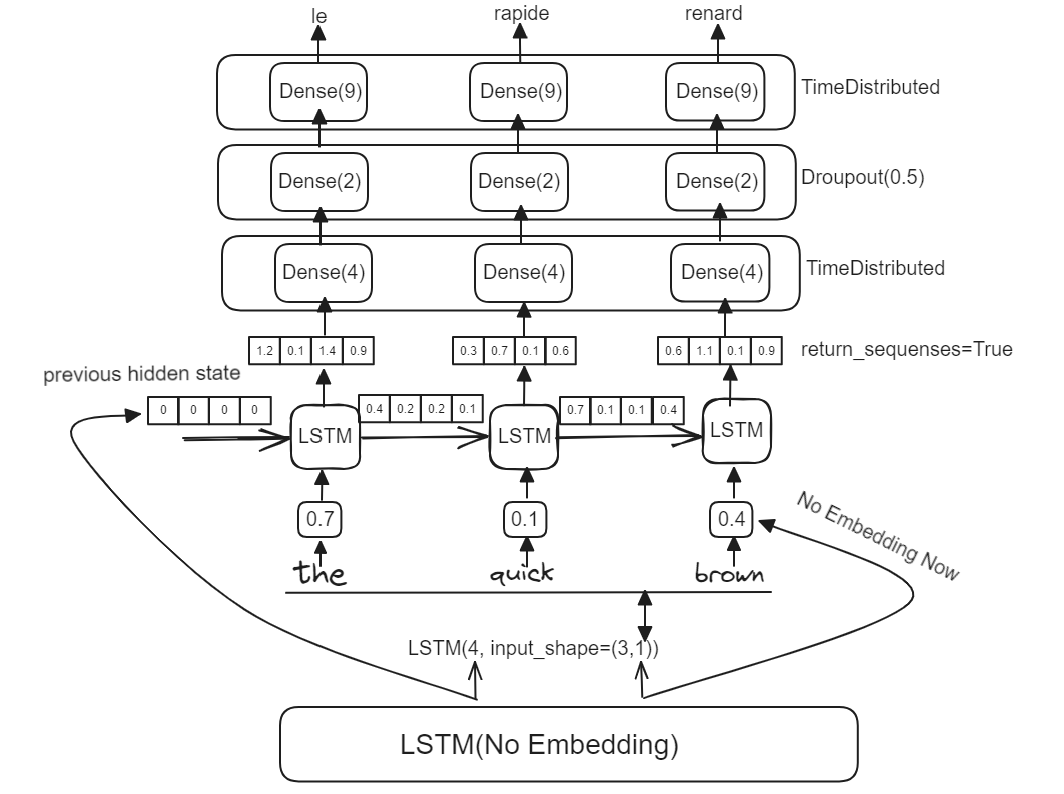
و input\_shape ثنائية الجزء الأول يضم عدد الكلمات في الجملة والجزء الأخر من الثنائية يمثل طول شعاع الكلمة وهو 1

1. طبقة SimpleRnn حيث تأخذ n\_neuros\_rnn الذي يمثل عدد الحالات المخفية بين الطبقات وتم وضع return\_sequences على القيمة true لكي يقوم كل عصبون بإرجاع hidden states الخاص به, الشكل (كذا) يوضح آلية الطبقة.
2. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN, وتم شرحها بشكل مفصل, الشكل (كذا) يوضح آلية الطبقة.
3. طبقة Dropout مع قيمة 0.5لمنع حصول over fitting في الشبكة, الشكل (كذا) يوضح آلية الطبقة.
4. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN, وتم شرحها بشكل مفصل, الشكل (كذا) يوضح آلية الطبقة.

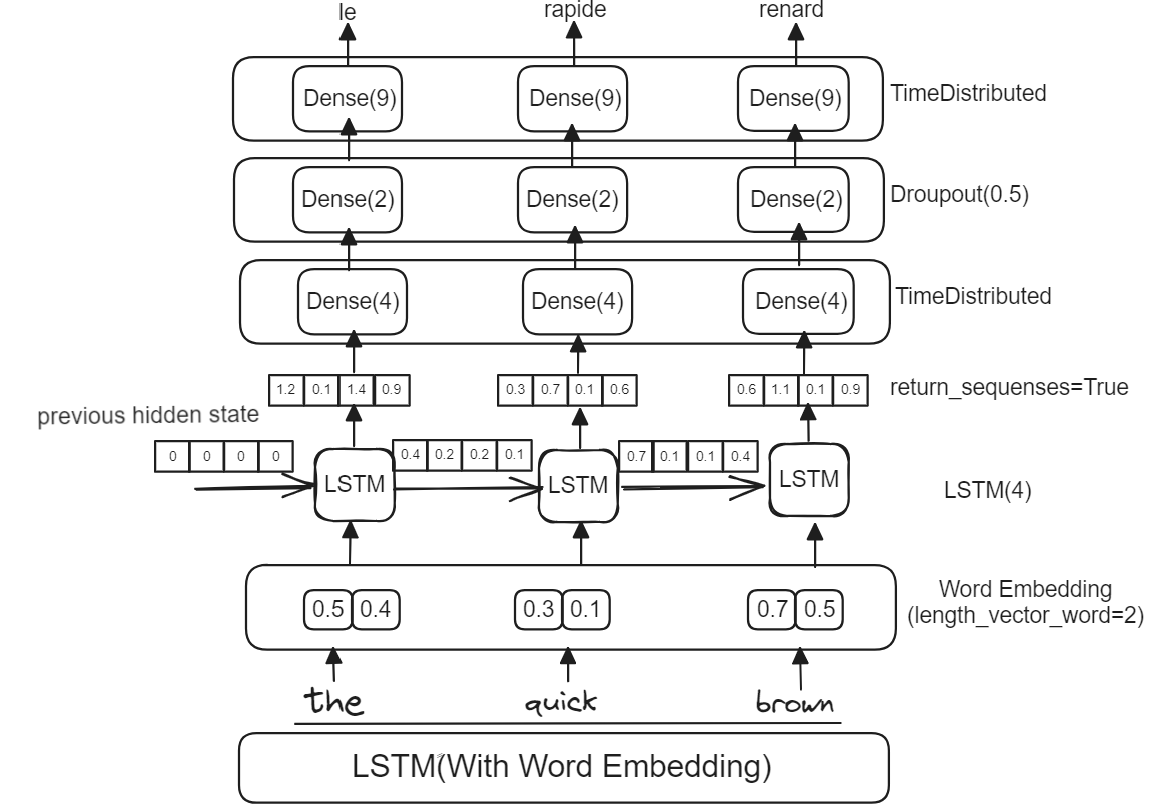


الشكل(كذا)

1. type\_model="Lstm" اذا الشبكة المستخدمة هي LSTM ولا يوجد طبقة  
   (Word Embedding), والطبقات على الترتيب التالي:
2. طبقة LSTM حيث تأخذ n\_neuros\_lstm الذي يمثل عدد الحالات المخفية بين الطبقات و input\_shape ثنائية الجزء الأول يضم عدد الكلمات في الجملة ومثل أيضا time steps عدد العصبونات الموجود في طبقة lstm والتي سوف يهتم كل عصبون فيها بكلمة واحدة من الجملة والجزء الأخر من الثنائية يمثل طول شعاع الكلمة و تم وضع return\_sequences على القيمة true لكي يقوم كل عصبون بإرجاع hidden states الخاص به, الشكل (كذا) يوضح آلية الطبقة.
3. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN, وتم شرحها بشكل مفصل, الشكل (كذا) يوضح آلية الطبقة.
4. طبقة Dropout مع قيمة 0.5لمنع حصول over fitting في الشبكة, الشكل (كذا) يوضح آلية الطبقة.
5. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN, وتم شرحها بشكل مفصل, الشكل (كذا) يوضح آلية الطبقة.

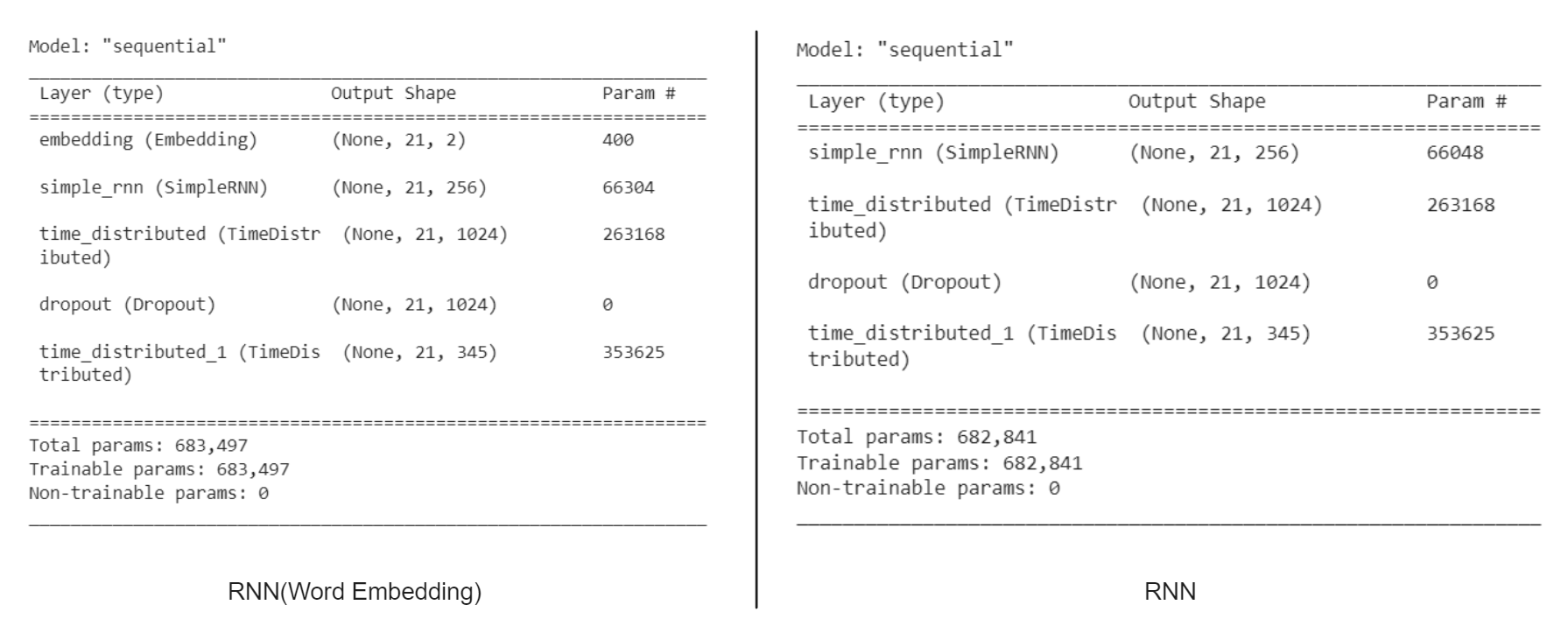
الشكل(كذا)

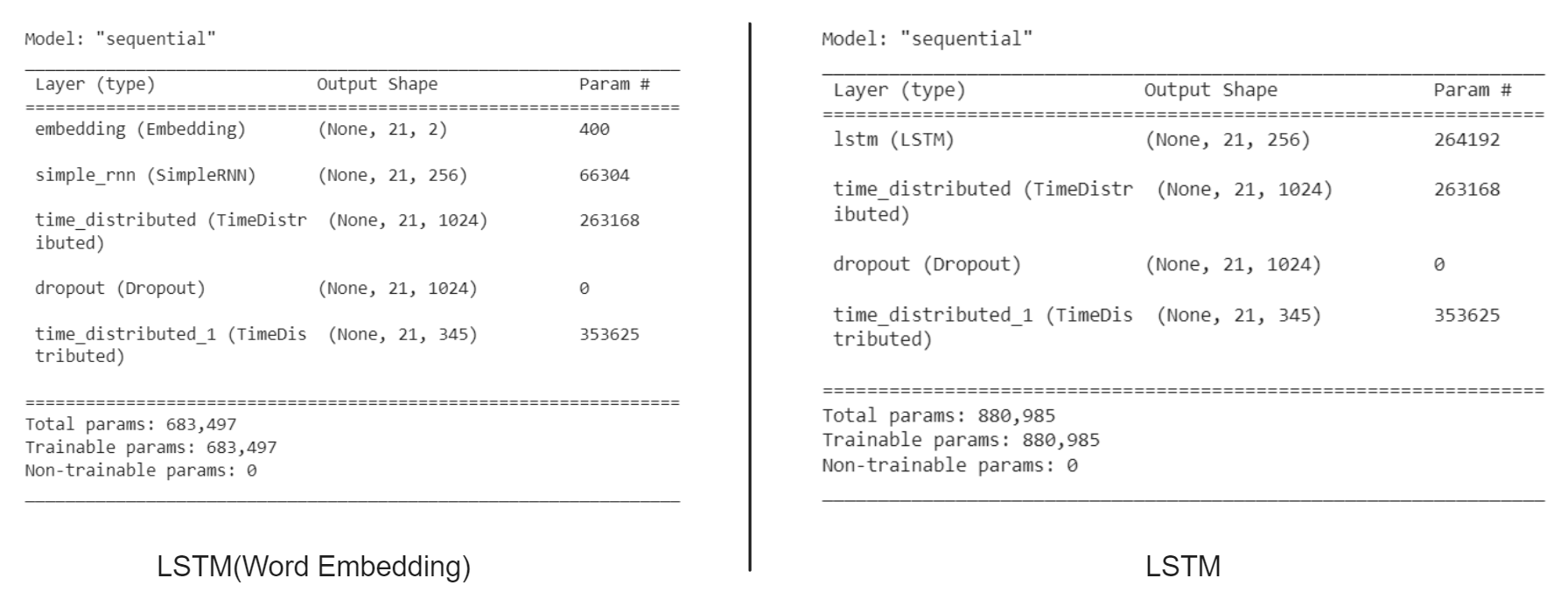
1. type\_model="Lstm\_Embd" اذا الشبكة المستخدمة هي LSTM مع طبقة  
   (Word Embedding), والطبقات على الترتيب التالي:
2. طبقة Embedding حيث تأخذ dict\_en\_sizeالذي يمثل عدد الكلمات الانجليزية المستخدمة في الداتا و length\_vector\_word تم شرحه سابقا في ملف train.yaml يمثل طول شعاع الكلمة الذي نريد الحصول عليه و input\_length يمثل طول أكبر جملة موجود في الداتا سواء جملة فرنسية أو انكليزية وهي 21 في حالتنا وinput\_shape ثنائية الجزء الأول يضم عدد الكلمات في الجملة والجزء الأخر من الثنائية يمثل طول شعاع الكلمة وهو 1
3. طبقة LSTM حيث تأخذ n\_neuros\_lstm الذي يمثل عدد الحالات المخفية بين الطبقات وتم وضع return\_sequences على القيمة true لكي يقوم كل عصبون بإرجاع hidden states الخاص به, الشكل (كذا) يوضح آلية الطبقة.
4. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN, وتم شرحها بشكل مفصل, الشكل (كذا) يوضح آلية الطبقة.
5. طبقة Dropout مع قيمة 0.5لمنع حصول over fitting في الشبكة, الشكل (كذا) يوضح آلية الطبقة.
6. طبقة TimeDistributed نفس آلية الطبقة في المعمارية RNN, وتم شرحها بشكل مفصل, الشكل (كذا) يوضح آلية الطبقة.



الشكل(كذا)

1. أيضا في التابع السابق تم استخدام تحديد يتم طباعة المعمارية التي تم استخدامها مع حجم وأوزان كل طبقة باستخدام التابع Summary المعرف في keras وكان المعماريات مع اسم كل معمارية مرافق ويوضحها الشكل (كذا).

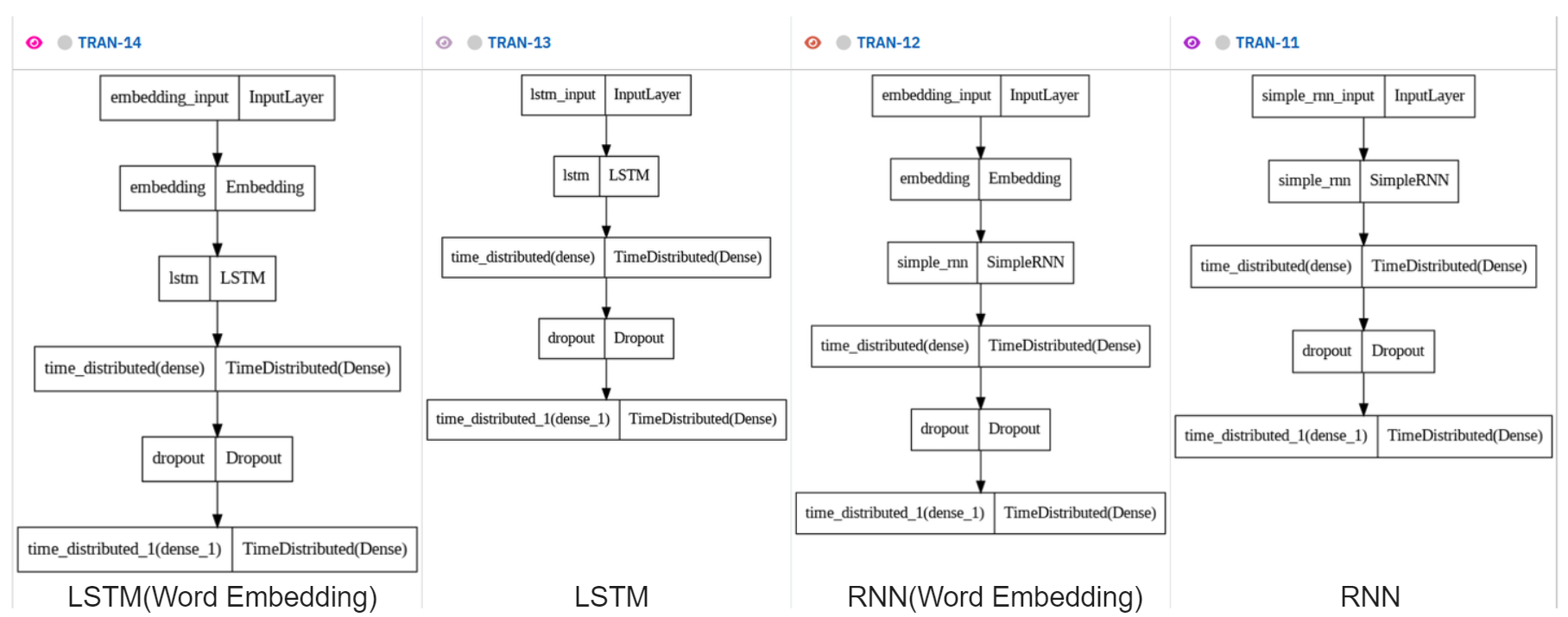


الشكل(كذا)

1. في التابع السابق تم استخدامcompile لتعريف توابع loss, metrics, optimizer ويقوم بعدها التابع motor بإرجاع model بعد كامل التجهيز, وبذلك نكون قد انتهينا من الحديث عن ملف models.py.

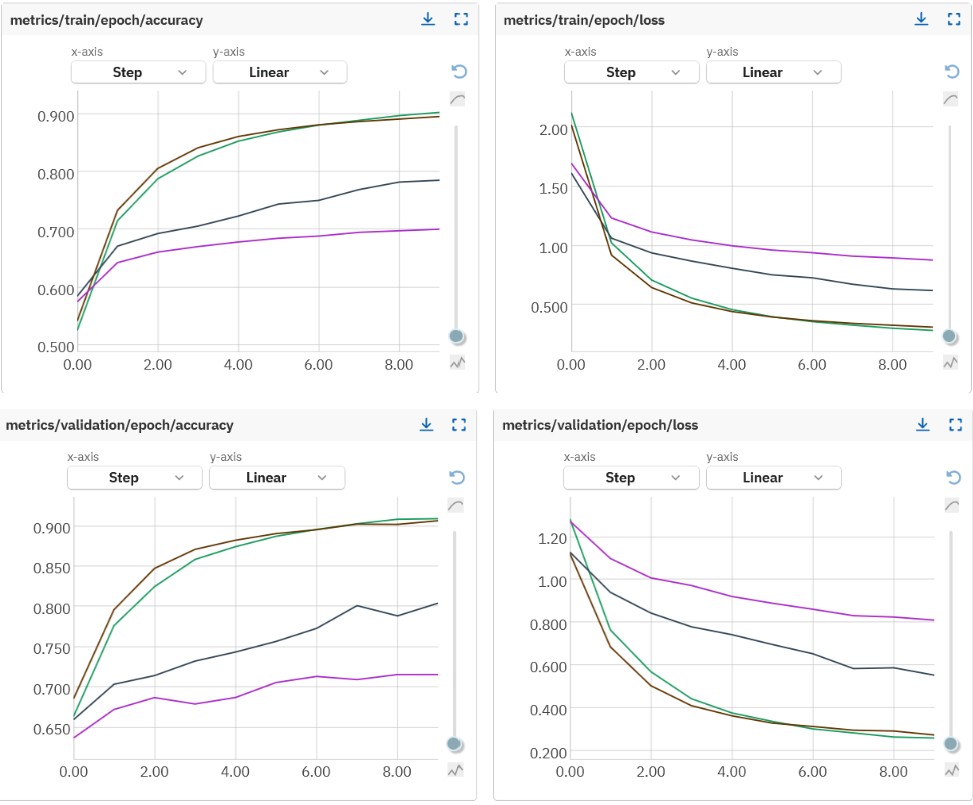
* في ملف train.py كما تحدثنا في مرحلة التخطيط أن هذا الملف سوف يستخدم في عملية التدريب, لذلك تم بناء التابع train الذي يأخذ متحول config\_path وهو مسار الملف train.yaml الذي يضم إعدادات عملية التدريب, بعد تعريف الإعدادات في ملف train.yaml وتعريف المعماريات في ملف models.py وتوابع المعالجة في ملف preprocess.py, يقوم التابع train بالتالي

1. قراءة جميع المتحولات في ملف train.yaml باستخدام المكتبة yamlوإسنادها إلى متحولات مطابقه لها بالاسم منعا من الضياع.
2. بدء مرحلة معالجة الداتا من خلال التوابع التي عرفناها في الملف preprocess.py, وذلك بقراءة الداتا المستخدمة على مرتين الانكليزية ثم الفرنسية باستخدام read\_data واستخدام التابع full\_process لتطبيق مرحلة tokenizer و padding.
3. قبل البدء بمرحلة التدريب تم تخزين مرحلتي المعالجة بما يخص مرحلة tokenizer لاستفادة من القاموس الخاص باللغتين في مرحلة التحول الكامل بعد التدريب وتم الحفظ على شكل ملفين tokenizer\_en.pkl و tokenizer\_fr.pkl.
4. الحصول على معمارية الشبكة باستخدام التابع motor عبر type\_model كما ذكرنا وكذلك تجهيز طبقات المعمارية المختارة باستخدام متحولات Structural Models
5. أما الآن سوف نتعرف على النوع الأول من monitoring في مشروعنا, حيث كان يجب عنه الحديث في مرحلة خاصة تحت مسمى **Performance Monitior**, ولكن منعا لتشت تناسق الأفكار الخاصة بالمشروع, يستخدم هذا النوع لجعل حياة Research أسهل وتقع عاتق توفير هذه المهمة على فريق MLOps, فهو يوفر طريقة لمراقبة عدة نماذج نقوم بتدريبها على التفرغ في مرحلة ومراقبة مخططات الخطأ و الكفاءة وكذلك مراقبة حالة CPU وGPU و Memory وبالتالي هذا يساعدنا على إطلاق الحكم على استخدام النموذج الأفضل بين النماذج للقيام بعملية تطويره لاحقا, لذلك سوف نقوم أداة Neptune.ai مع keras, قمنا باستخدام التابع init\_run في neptune لتجهيز المشروع باستخدام project="ammar.mlops/translate-en-to-fr" ويوجد متحول أخر name يأخذ قيمة type\_model الحالية تكمن فائدة هذا المتحول في تسمية كل عملية تدريب وبالتالي ف Rnn لها نتائج مستقلة عن RNN\_Embd هذا التنظيم يساعد في التتبع الجيد لعمل كل model ضمن المشروع الواحد والمتحول الاخير api\_token يمكن اعتباره بمثابة باسورد تعطي سماحية لمستخدمها من الولوج إلى موقع neptune وتخزين نتائج تدريب النماذج عليه, بعد ذلك تم ربط neptune مع keras تعريف callbacks حيث يقوم هذا المتحول بكتابة كل عملية التدريب من قيم الخطأ و الكفاءه ورسم مخططات في موقع neptune.
6. بعد ذلك يتم استخدام GPU أو CPU عبر المتحول type\_device, وبداية عملية التدريب عبر التابع fit المعرف في keras والذي يأخذ متحولات تم شرحها بالتفصيل سابقا خلال المراحل السابقة مثل process\_input و process\_output وbatch\_size و epochs و validation\_split و shuffle لكي تصبح الداتا مرتبة عشوائيا في كل عملية تدريب واخير callbacks الذي عرفناه مسبقا لتحصل عملية الربط بين neptune وkeras
7. تم استخدام التابع save في keras لحفظ model وتم استخدام type\_model ليختلف اسم النموذج المدرب وبالتالي يسهل التعرف عليه واستخدامه.
8. تم كتابة train.py على شكل script وهذا ما يناسب بيئة التدريب عند العمل على سيرفر مستقل أو colab بحيث يكون هذا الملف قابل للعمل في أي مكان يوضح فيه ويتم تنفيذه بالشكل التالي وتسمح لنا المكتبة argparse بإعطاءه config\_path كمتحول خارجي والتنفيذ يكون على الشكل التالي:  
   
9. بعد التنفيذ دخلنا إلى موقع neptune.ai وقمنا بمراقبة عمل النماذج على التوازي وكذلك مراقبة مخططات الخطأ والكفاءة وحصلنا على النتائج التالية, في البداية نستعرض مخططات المعماريات على التوازي ويوضحها الشكل(كذا)



الشكل(كذا)

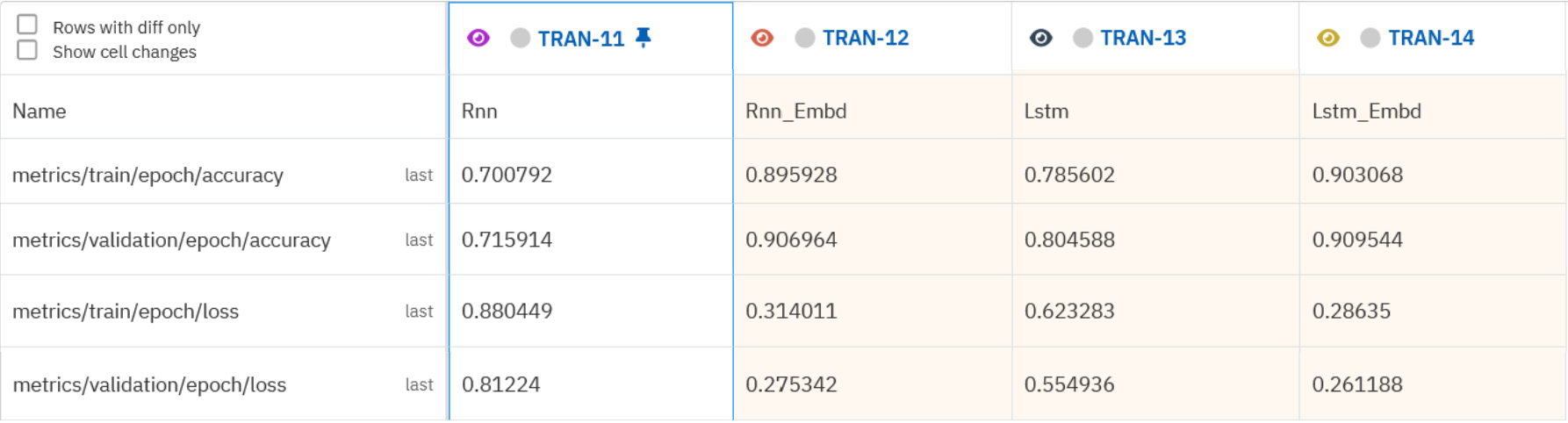
وكذلك مخططات loss و accuracy في كل epochs بالنسبة للأربع معماريات حيث نلاحظ أن بالنسبة لمخطط accuracy في الشكل (كذا) على اليسار سواء train أو validation أن الخطين البيانين في أعلى كل مخطط يحققان accuracy أعلى خلال زيادة قيمة epochs فيما الخطين البيانين في أسفل كل مخطط يحققان accuracy أقل وهذا يدل على أن النماذج تتعلم أفضل سواء LSTM أو RNN عند استخدام طبقة Word Embedding)) وكذلك مخطط loss في الشكل(كذا) على اليسار سواء train أو validation تعطي نفس الحقيقة حيث الخطين في أسفل كل مخطط يحققان loss أقل عند تقدم epochs وبالتالي طبقة Word Embedding)) حققت فرق ملموس وواضح بين المعماريات التي تستخدم هذه الطبقة والمعماريات التي لا تستخدمها



الشكل(كذا)

**النتائج**

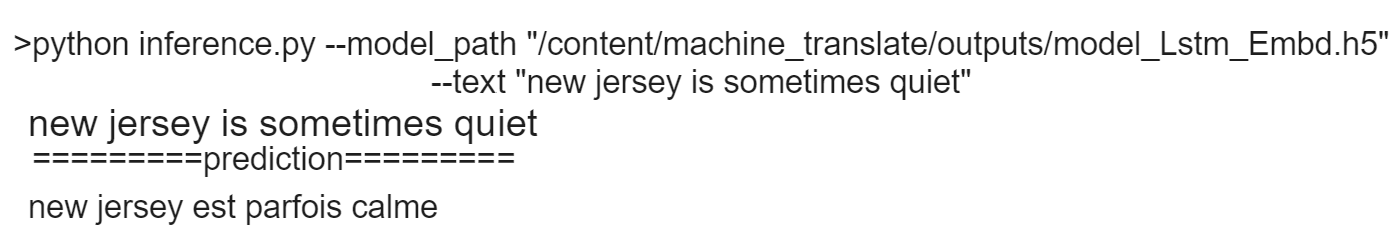
من خلال عملية الربط بين neptune وkeras نستطيع الحصول على جدول لنتائج المقارنة بين الأربع معماريات الشكل(كذا), لن نستعرض كامل الجدول سوف نستعرض ما يهمنا من عملية التدريب في الأربع معماريات وأي معمارية أفضل بالنسبة للنتائج الحالية , حيث على الترتيب الجدول على مستوى الأعمدة كل عمود له اسم يمثل رقم فريد للمعمارية المستخدمة, على مستوى الاسطر في السطر الأول اسم النموذج المدرب وبعدها قيمة الكفاءة بالنسبة ل train وvalidation وقيمة الخطأ بالنسبة train وvalidation نلاحظ نفس الحقيقة التي وجدناها في المخططات نجدها في الجدول الشكل(كذا), المعماريات التي تستخدم Word Embedding)) أي العمود ذات الاسم tran12 و tran14أفضل في عملية التعلم وبالتالي سوف تكون أفضل في عملية التحول الكامل



الشكل(كذا)

بعد ذلك يتم اختبار النموذج بعد عملية التدريب ويتم ذلك عن طريق الملف inference.py

حيث يقوم هذا الملف بتحويل جملة من اللغة الانكليزية إلى الفرنسية بعد اختيار احد النماذج الاربعة التي تم تدريبها وإعطاءه رابط النموذج وكذلك تمرير الجملة المراد ترجمتها, فيقوم بتحميل الاوزان الموجودة في مجلد ,outputs طبعا مكتبة argparse هي التي تسمح لنا بهذا النوع من العمليات, بعد تنفيذ الملف نلاحظ الخرج الذي يمثل الجملة المراد ترجمتها والترجمة المتوقعة.



إلى هنا نكون قد انتهينا من مرحلة بناء وتدريب النماذج Build and Train Models وسوف ننتقل إلى المرحلة الرابعة الخاصة بمرحلة التعبئة (packaging)

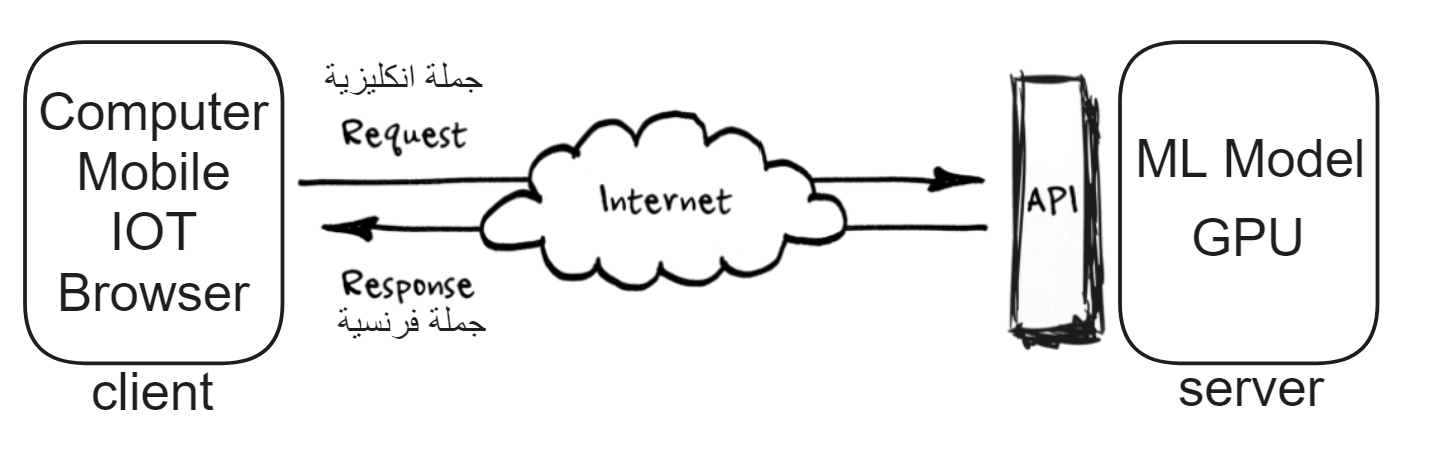
**المرحلة الرابعة: التعبئة packaging**

في هذه المرحلة يتم تجميع جميع المكونات والتبعيات اللازمة لتشغيل تطبيق أو نموذج ذكاء صناعي في حزمة مستقلة. وفي حالتنا سوف نقوم باستخدام fastapi وذلك لتعريف API تقوم بتحميل نموذج وتنفيذ الترجمة واستخدام Docker لتعريف البيئة المعزولة التي ستحتوي على جميع التبعيات و الأدوات اللازمة لتشغيل النموذج, وبالتالي لدينا مرحلتين هما:

* **مرحلة تحويل ML Model إلى API:**

ما هوAPI؟ هو مجموعة من البرمجيات والقواعد والبروتوكولات التي تسمح للتطبيقات بالتواصل والتفاعل مع بعضها البعض, يعتبر API نقطة اتصال بين تطبيقين مختلفين أو بين تطبيق وخدمة خارجية. يستخدم التطبيق المصدر (client) API للوصول إلى وظائف أو بيانات تقدمها التطبيق المستهدف (server), الشكل(كذا) يشرح الآلية حيث يتم إرسال الجملة الانكليزية بصيغة json مثلا عبر الأنترنت إلى API حيث يقوم بمعالجة الطلب على server ومن ثم يقوم API بإعادة الجملة الفرنسية بصيغة json أيضا عبر الأنترنت إلى client من الممكن أن يكون client أي جهاز يملك الحد الأدنى لوصف ألة تعمل بالكهرباء وتملك معالج وذاكرة ويمكنها الاتصال بالانترنت مثل

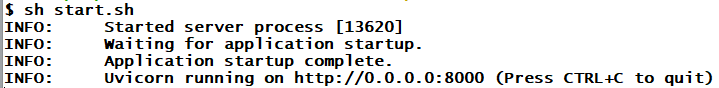
computer, mobile, sensor..etc.



الشكل(كذا)

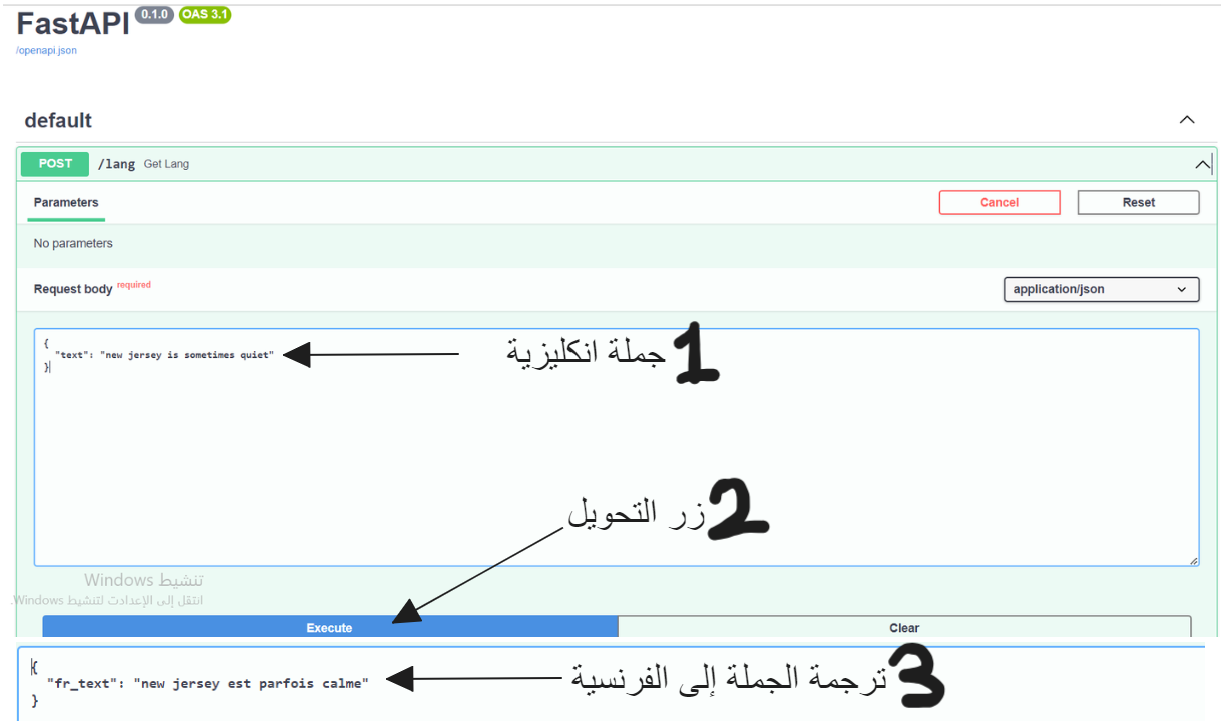
بعد شرح API ننتقل الأن إلى تقديم شرح عن دمج ml model مع fastapi وهو كما ذكرنا سابقا إطار عمل لبناء API ويتم ذلك في مجلد fastapi والذي يحوي:

1. المجلد الأول www يملك ملف init\_fastapi.py يقوم بأخذ object من FastAPI الذي يمثل تطبيق منفصل وذلك لتعريف المسارات والوظائف المرتبطة بهذا التطبيق حيث في تطبيقنا لدينا وظيفة واحدة وهي تحويل الجملة من الانكليزية إلى الفرنسية.
2. المجلد الثاني routes في هذا المجلد يتم تعريف المسارات لاستقبال الطلبات ((requestsواستخدام التوابع المناسبة لمعالجة الطلب على server , وفي ملف lang.py قمنا ببناء التابع lang حيث تستقبل english\_text مرسل من قبل client وتقوم باستخدام التابع en\_to\_fr لمعالجة الطلب والذي يتم في المجلد الثالث handlers ثم تقوم بإرجاع الجملة الفرنسية إلى client.
3. المجلد الثالث handlers وهو المجلد الذي يقوم بمعالجة الطلب على server, حيث في ملف lang.py ضمن المجلد نقوم بتحميل الأوزان الخاصة بالمعالجة مثل tokenizer\_en.pkl, tokenizer\_fr.pkl والأوزان الخاصة بمرحلة التدريب طبعا تم اختيار النموذج الأكبر دقة وهو LSTM(Word Embedding) وبعد ذلك تم تعريف التابع en\_to\_fr الذي يقوم باستخدام tokenizer ثم padding ثم predict المعرفة في keras ثم تحويل الأرقام كلمات باللغة الفرنسية ثم إعادة الناتج.
4. ملف server.py والذي يتم فيه تعريف IP address + Port Number لأنشاء عملية الاتصال عندما يريد client التخاطب مع FastAPI server.
5. ملف start.sh لبدء عملية تشغيل .server

بعد تشغيل السيرفر باستخدام الملف الاخير نلاحظ الخرج التالي وهو يعني جاهزية السيرفر لاستقبال طلبات الترجمة على الرابط التالي للاستخدام المحلي http://127.0.0.1:8000   


نستطيع إرسال طلب عبر استخدام command line ولكن احد اسباب اختيار FastAPI هو أنه يقوم بتوفير Swagger UI تريحنا من هذه العملية وتوفر لنا واجهة مستخدم سهلة للتعامل معها ويتم ذلك عن استخدام نفس الرابط مع إضافة كلمة docs بالشكل http://127.0.0.1:8000/docs

فيظهر لنا هذه الواجهة مع كامل عملية التحويل الشكل(كذا)



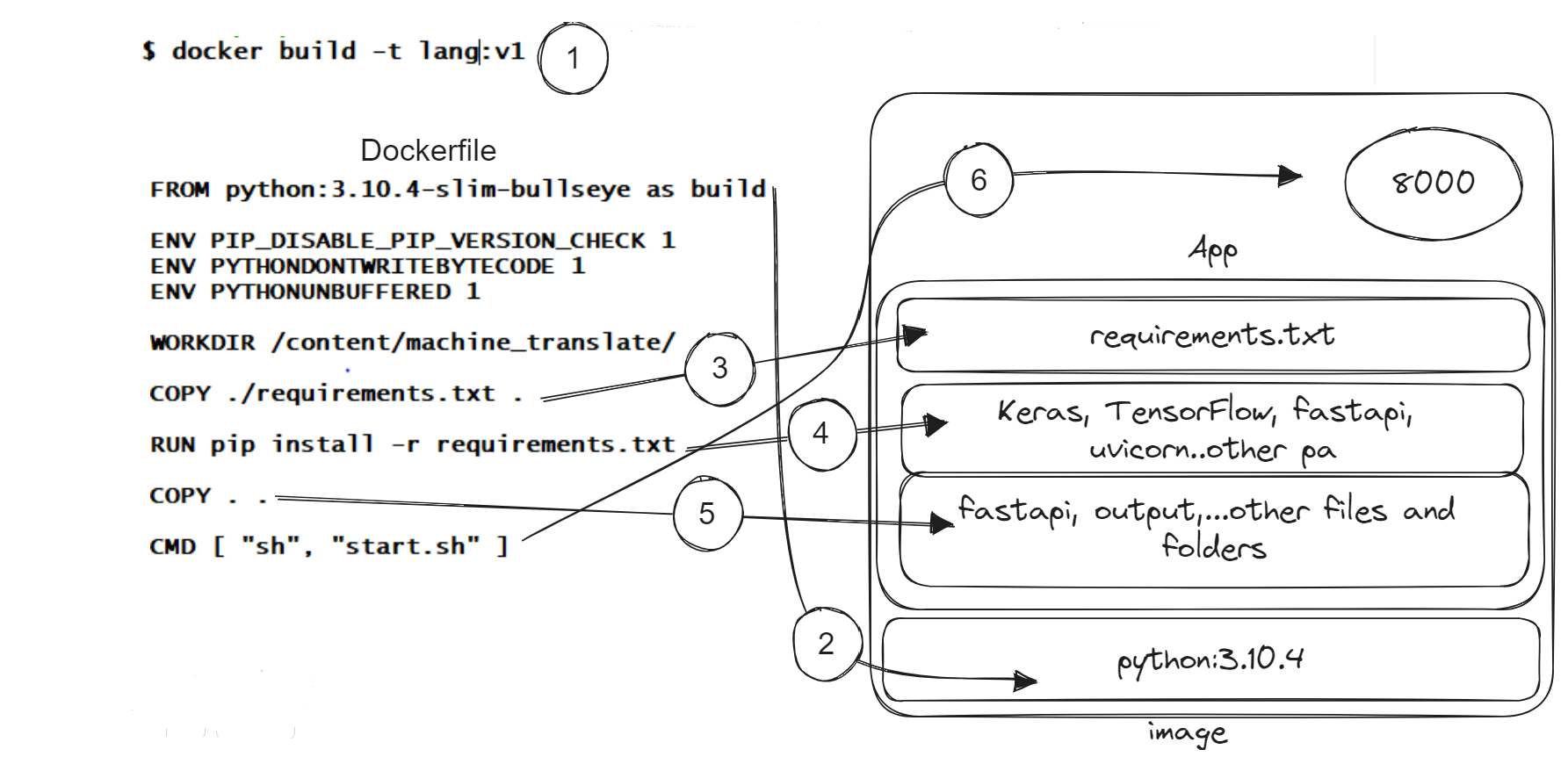
الشكل(كذا)

* **مرحلة تشغيل API في بيئة معزولة باستخدام Docker:**

ماهو Docker؟ Docker هو منصة تقنية تسمح بتعبئة وتشغيل التطبيقات والخدمات داخل بيئة معزولة عن النظام المضيف. يستخدم Docker تقنية الحاويات (containers) لتوفير بيئة مستقلة وقابلة للتكرار لتشغيل التطبيقات.

ببساطة، يمكنك اعتبار Docker كصندوق يحتوي على جميع المكونات والأدوات اللازمة لتشغيل تطبيق معين. هذا يشمل الأكواد والتبعيات والمكتبات والأدوات وأي شيء آخر يحتاجه التطبيق للعمل بشكل صحيح, عند استخدام Docker يمكنك إنشاء صورة Docker تحتوي على جميع المكونات اللازمة لتشغيل التطبيق. يمكن تشغيل هذه الصورة على أي جهاز يدعم Docker، بغض النظر عن نظام التشغيل الذي يستخدمه الجهاز, والتي توفر بيئة معزولة تمامًا عن النظام المضيف.

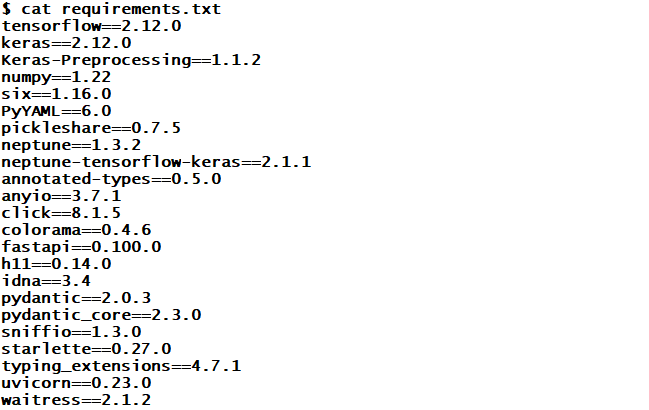
باستخدام Docker، يمكنك تعبئة التطبيق بسهولة وتشغيله على أي جهاز بدون الحاجة لتثبيت المكونات والتبعيات يدويًا. كما يمكنك توزيع التطبيق كحزمة مستقلة، مما يسهل على المستخدمين تشغيله واستخدامه بسهولة. وللقيام بهذه المهمة سوف نقوم بكتابة ملف Dockerfile يحوي بعض التعليمات لبناء هذا الصندوق الذي يوضحه الشكل (كذا) بشكل تخيلي وسنوضح في الأسطر اللاحقة أرقام المراحل



الشكل(كذا)

عند بناء هذه الصورة (الصندوق) أحد مفاهيم docker يتم اعتبار كل أمر سطري في ملف dockerfile طبقة وتستطيع كل طبقة أن تستفيد من الطبقة التي تسبقها, دون الدخول في تفاصيل كثيرة لنبدأ في شرح المراحل المرقمة بالترتيب في الشكل(كذا):

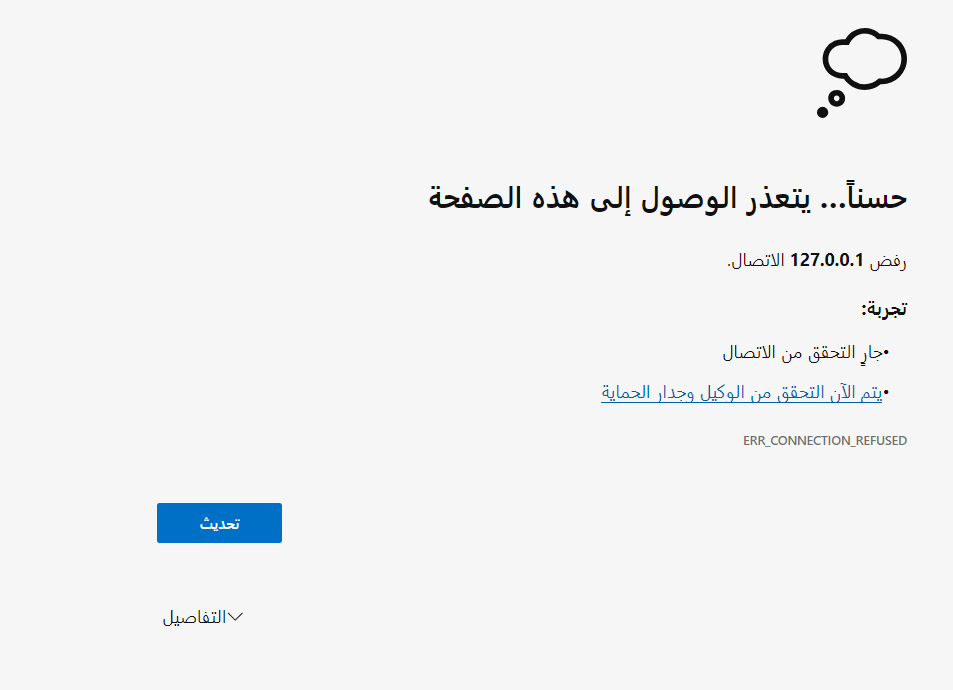
1. بعد كتابة Dockerfile على اليسار وتنفيذ الأمر في command line والذي يمثل عملية بناء الصورة يتم بناء الصورة التخيلية على اليمين ونستطيع اعطاء اسم لهذه الصورة مع رقم يمثل الإصدار عبر المتحول (-t) وهذا يساعد في الرجوع إلى اصدارات سابقة, تم تسمية الاصدار lang:v1.
2. استخدام الطبقة ضمن الصورة (الصندوق) التي تسمح بتشغيل أي تطبيق بايثون مع تحديد رقم اصدار, تم اختيار python:3.10.4-slim-bullseye وهو اصدار خفيف من الاصدار العادي وذلك لتوفير الذاكرة, طبعا لو أردنا تشغيل تطبيق php كنا استخدمنا صورة خاصة ب php.
3. نقل الملف requirements.txt ضمن جهازنا المضيف إلى داخل الصورة (الصندوق) وذكرنا في مرحلة التخطيط أن هذا الملف يحوى على python packages الخاصة بالمشروع فقط مع رقم الأصدار لكل packages



1. تنفيذ أمر تثبيت الحزم بعد نقل الملف في المرحلة 3 عبر الأمر pip install
2. نقل مجلدات وملفات المشروع ضمن جهازنا المضيف إلى داخل الصورة (الصندوق)
3. وضع أمر تشغيل السيرفر الخاص بAPI في لحظة تحويل هذا الصورة (الصندوق) إلى Container عبر port 8000.

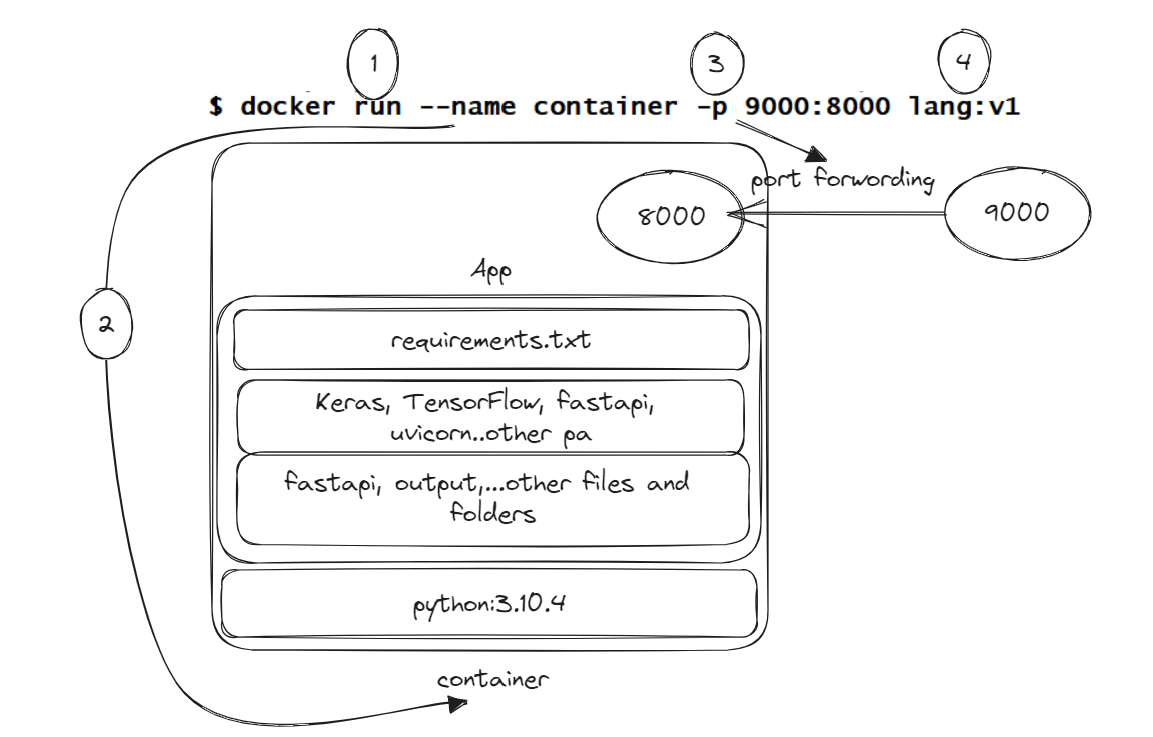
ماهي Container ؟هي تعتبر المرحلة التشغيلية للمشروع والاستفادة من خدماته أي تحول هذا الصندوق الذي يحوي جميع الأشياء السابقة إلى صندوق قابل للاستخدام, بكلمات أخرى Container أصبح سيرفر صغير ضمن جهازنا يعمل عليه تطبيق الترجمة ويستطيع هذا السيرفر الاستفاده من ram وcpu وتشغيل تطبيق الترجمة بسبب الطبقات الموجودة لديه والتي عرفناها في الصورة (الصندوق) في ملف Dockerfile ,الفرق بين image وcontainer مثل الفرق بين الملف المضغوط والملف بعد فك الضغط في الملف المضغوط لا تستطيع التعديل على أي من ملفاته وكذلك الصورة بعد الكتابة عليها وبناءها اصبحت مثل الملف المضغوط لا يمكن التعديل عليها ولا ان نستفيد من خدماتها لذلك نقوم بتحويلها إلى Container بالتالي تصبح كأنه ملف بعد فك الضغط نستطيع حذف ملفاته والتعديل عليها وكتابتها وكذلك Container أصبحنا باستطاعتنا أن نستفيد من الخدمات التي تقدمها وهي الترجمة في حالتنا ويكون لها ip وport الذي حددناه مسبقا عند بناء الصورة ويساوي 8000, طبعا هنا ستقابلنا مشكلة لو قمنا بتشغيل container ودخلنا إلى الرابط http://127.0.0.1:8000

سوف يعطنا هذا الخطأ في الشكل (كذا)



الشكل(كذا)

لذلك سوف نستخدم مفهوم في الشبكات يدعىport forwording وهو يقوم على ربط port من النظام الخاص بنا(وهو هنا الحاسب الخاص بنا) مع port لجهاز أخر(وهو في حالتنا container الذي يملك port 8000) وذلك عند تشغيل container الموضح في الشكل(كذا)

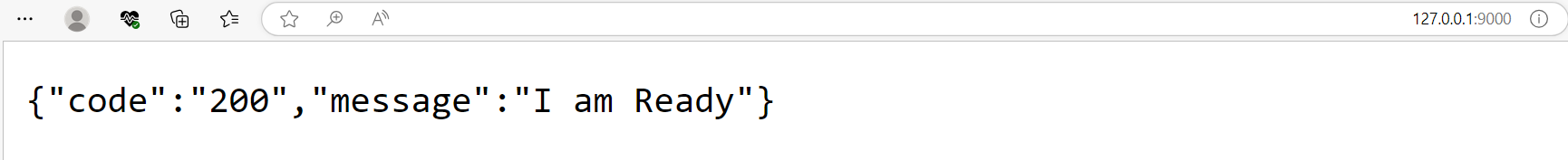


الشكل(كذا)

حيث تعني المراحل الموضحة في الشكل (كذا):

1. استخدام أمر التشغيل run لأخذ نسخة من الصورة وتحويلها إلى Container.
2. اعطاء اسم للحاوية وهنا تم تسميتها container.
3. تكنيك port forwarding يسمح للبورت 9000 في الجهاز المضيف بالاستماع إلى البورت 8000 داخل container بالتالي عند فتح الرابط http://127.0.0.1:9000

نلاحظ نجاح العملية الشكل(كذا).



1. استخدام الصورة مع اصدارها والذي قمنا ببناها سابقا lang:v1.

إلى هنا نكون قد انتهينا من مرحلة التعبئة (packaging) وسوف ننتقل إلى المرحلة الرابعة الخاصة بمرحلة deployment

**المرحلة الرابعة: مرحلة المنتج النهائي Deployment**

**المرحلة الرابعة: الاختبارTest**

في هذه المرحلة لن نتطرق إلى كافة عمليات الاختبار ولكن سنذكر المراحل الاساسية في عمليات الاختبار:

* ذكرنا في مرحلة التخطيط وجود بيئة (Train) خاصة بتدريب النماذج, بيئة((dev خاصة بتحويل أوزان النماذج إلى web service)) والبيئة النهاية (prod) خاصة بإطلاق web service إلى مرحلة الاستخدام من قبل المستخدم ((in real world وبالتالي يجب أن يكون لكل بيئة عمليات اختبار خاصة بها وبالتالي يجب بناء unit test لكل بيئة بحيث تتم هذه العملية بشكل مؤتمت automation دون تدخل من قبل المبرمج .
* في بيئة train يجب أن تكون عمليات test موجودة قبل عملية التدريب:

1. اختبار توابع المعالجة الموجودة في ملف preprocess.py وتم ذلك عن طريق test\_preprocess.py وتم شرح الملفين في مرحلة المعالجة.
2. اختبار المعماريات الموجودة في ملف models.py.

اختبار خرج النموذج بعد عملية التدريب ويتم ذلك عن طريق الملف inference.py

حيث يقوم هذا الملف بتحويل جملة من اللغة الانكليزية إلى الفرنسية بعد اختيار احد النماذج الاربعة التي تم تدريبها وإعطاءه رابط النموذج وكذلك تمرير الجملة المراد ترجمتها, فيقوم بتحميل الاوزان الموجودة في مجلد outputs طبعا مكتبة argparse هي التي تسمح لنا بهذا النوع من العمليات, بعد تنفيذ الملف نلاحظ الخرج الذي يمثل الجملة المراد ترجمتها والترجمة المتوقعة.

