

Getting Started Guide: *upsay_jupyter_ai* toolkit

Paulo Henrique Couto¹

¹Université Paris-Saclay, France

<https://mydocker.universite-paris-saclay.fr/shell/join/upsayai>

<https://gitlab.dsi.universite-paris-saclay.fr/upsayai>

https://pypi.org/project/upsay_jupyter_ai

November 13, 2024

1 Initialization

In order to provide flexibility to users during a first testing period, the toolkit currently requires the initialization as an object of the class `UPSayAI`, from the `upsay_jupyter_ai` python package, where several parameters can be configured:

```
from upsay_jupyter_ai import UPSayAI

UPSayAI(model_name="aristote/llama",
        model_access_token=my_access_token,
        model_temperature=0.8,
        model_max_tokens=2048,
        rag_model="dpr",
        rag_min_relevance=0.5,
        rag_max_results=5,
        recommendation_min_relevance=0.7,
        quiz_difficulty="moyen",
        quiz_min_relevance=0.2,
        quiz_max_results=100,
        num_questions_quiz=5,
        course_corpus_file="corpus_ISD.csv");
```

It is worth noticing that the object itself is not used, this procedure is only used to be able to configure the parameters of the toolkit at the start of the session before using the commands. Each parameter has the default value shown in the snippet above. Therefore, for now, for the project developers who are currently testing the prototype, the only mandatory argument to pass is the access token (represented by the variable `my_access_token` in the snippet above), whether from Hugging Face for the “huggingface/llama” model, or from Aristote Dispatcher for the “aristote/llama” and “aristote/mixtral” models. Note that, for now, the **ISD Corpus CSV file** must be uploaded to the **MyDocker** environment so that the toolkit can access it. Additionally, as the current **MyDocker** environment does not have GPUs, the use of “huggingface/llama” model is not possible for the moment.

All the parameters are described below:


- **model_name:** Model name among three possible alternatives: “huggingface/llama”, which represents the model meta-llama/Meta-Llama-3-8B-Instruct imported from Hugging Face; and

“aristote/llama” and “aristote/mixtral”, which represent the casperhansen/llama-3-70b-instruct-awq and the casperhansen/mixtral-instruct-awq models, respectively, stored in the Aristote Dispatcher;

- **model_access_token:** Access token to use the LLM selected by the model_name parameter. In the case of “huggingface/llama”, it should be a Hugging Face access token, whereas for either “aristote/llama” or “aristote/mixtral”, it should be an Aristote Dispatcher access token. For now, the user must initialize the toolkit using a valid individual token;
- **model_temperature:** LLM’s temperature parameter for generation, which determines its creativity. It must be a float between 0 and 1;
- **model_max_tokens:** LLM’s max tokens parameter for generation. In the case of “huggingface/llama”, it is used as max_new_tokens instead, since the model does not expect a max_tokens parameters. It must be an int between 512 and 4096;
- **rag_model:** IR model to be used by the RAG logic. Current options are “dpr”, which uses the cosine similarity of dense embeddings generated by the AgentPublic/dpr-question_encoder-fr_qa-camembert model available in Hugging Face, and “bm25”, which uses Okapi BM25;
- **rag_min_relevance:** Minimum relevance threshold for the cosine similarity in the RAG logic when using “dpr” as rag_model. It must be a float between 0 and 1;
- **rag_max_results:** Maximum number of extracts (in this case, Jupyter cells) to be passed to the LLM as context by the RAG logic. Used both by the “dpr” and “bm25” algorithms. It must be an int between 1 and 100;
- **recommendation_min_relevance:** Minimum relevance threshold for the cosine similarity in the recommendation system, which suggests notebooks to be reviewed by the students based on their questions. It must be a float between 0 and 1;
- **quiz_difficulty:** Level of difficulty of the questions generated for the quiz. It must be either “facile” for an easy level, “moyen” for an intermediate level, or “difficile” for a difficult level;
- **quiz_min_relevance:** Minimum relevance threshold for the cosine similarity in the RAG logic using “dpr” model to provide relevant extracts to the quiz generation. This is only used if the quiz is generated about a specific subject rather than about a specific notebook, since in this case the LLM will receive relevant passages from the classes to use as reference to create the questions. It must be a float between 0 and 1;
- **quiz_max_results:** Maximum number of extracts (in this case, Jupyter cells) to be passed to the LLM as context by the RAG logic to generate the quiz. This is only used if the quiz is generated about a specific subject rather than about a specific notebook. It must be an int between 10 and 1000;
- **num_questions_quiz:** Number of questions to be generated by the quiz. Note that, every time the LLM fails to correctly format a question (python dictionary), it is automatically deleted and the number of questions is reduced by one. Therefore it is possible that a quiz is generated with fewer questions that expected. It must be an int between 3 and 10;
- **course_corpus_file:** CSV file with all the course corpus (jupyter cells) for RAG and recommendation system. It contains every cell’s content, alongside their tokenized version, for “bm25”, and pre-calculated dense embeddings, for “dpr”. Note that, for now, the **ISD Corpus CSV file** must be uploaded to the **MyDocker** environment so that the toolkit can access it.

2 Using the *%ai_help* command

The *%ai_help* command is designed to provide a quick user guide that briefly explains each one of the commands available in the toolkit. To use it, the user simply needs to place “*%ai_help*” in an empty code cell and run it. The user guide screenshot in [Figure 1](#) illustrates the experience within a Jupyter Notebook when using the command.

UPSay AI Magic Commands chargé avec succès 

Cette boîte à outils est expérimentale et en phase de test. Elle peut donc présenter des anomalies.

Utilisez `%ai_help` pour accéder au Guide d'Utilisation.

[4]: `%ai_help`

Guide d'Utilisation

L'UPSay AI Magic Commands est un ensemble d'outils expérimentaux d'IA Générative inspiré de jupyter-ai et actuellement en cours de développement à l'Université Paris-Saclay.

Liste des Commandes Magiques :

`%ai_question`

Description : Commande magique conçue pour les questions/réponses (Q&R).

Mode d'emploi : Placez la commande dans la première ligne d'une cellule, suivie d'une question sur la même ligne. La réponse apparaîtra au format Markdown juste en dessous de la cellule.

Exemple d'utilisation :

[42]: `%ai_question` Quelle est la différence entre causalité et corrélation ?

Note : Vous pouvez utiliser `%%ai_question` (double %) si vous préférez utiliser plusieurs lignes pour formuler votre question.

`%ai_code`

Description : Commande magique conçue pour la génération de code (python).

Mode d'emploi : Placez la commande dans la première ligne d'une cellule, suivie des instructions pour la génération du code sur la même ligne. Le code généré apparaîtra dans une nouvelle cellule.

Exemple d'utilisation :

[42]: `%ai_code` Fonction qui calcule le déterminant d'une matrice numpy.

Note : Vous pouvez utiliser `%%ai_code` (double %) si vous préférez utiliser plusieurs lignes pour formuler vos intructions de code.

Figure 1: User guide invoked by the *%ai_help* command.

3 Using the *%ai_question* command

The *%ai_question* command is designed for questions and answers (Q&A). To use the *%ai_question* command, the user simply needs to place “*%ai_question*” at the beginning of an empty code cell, followed by the question on the same line, and run it. Note that the toolkit has access to a history of interactions made during the current session, so the user may ask follow-up questions in a chat-like conversation, always using the same formatting explained above (i.e. using a new code cell or replacing

the question in the same cell and re-running it). The screenshot in [Figure 2](#) illustrates the use of the command.

```
[19]: %ai_question Quelle est la différence entre causalité et corrélation ?
```

Excellente question !

La différence entre causalité et corrélation est fondamentale en science des données. Vous avez déjà vu ce contenu pendant le cours, mais voici un rappel utile.

La **corrélation** décrit une relation statistique entre deux variables, où une variable change en fonction de l'autre. Il existe trois types de relations statistiques : corrélation positive, corrélation négative et absence de corrélation. Par exemple, la figure qui montre la relation entre la consommation de chocolat et le nombre de prix Nobel illustre une corrélation positive. Cependant, cela ne signifie pas que consommer du chocolat conduit à gagner un prix Nobel !

D'un autre côté, la **causalité** décrit une relation de cause à effet entre deux variables, où une variable est la cause directe de l'autre. Dans l'exemple du chocolat et des prix Nobel, la richesse économique est la cause qui implique à la fois une augmentation de la consommation de chocolat et une augmentation du nombre de prix Nobel. La corrélation positive entre les deux variables n'implique pas une causalité directe.

En résumé, la corrélation décrit une observation, tandis que la causalité relève de l'interprétation. La corrélation peut être due à d'autres facteurs, comme la richesse économique dans notre exemple. Il est important de ne pas confondre ces deux concepts pour éviter des erreurs d'interprétation.

Pour résumer :

- La corrélation décrit une relation statistique observée entre deux variables.
- La causalité décrit une relation de cause à effet entre deux variables.

Il est essentiel de distinguer ces deux concepts pour éviter des conclusions erronées dans nos analyses de données.

💡 Pour plus d'informations sur ce sujet, il peut être utile de cliquer sur les liens pour réviser les cours : [CM2](#)

⚠️ Réponse générée par une intelligence artificielle, les informations peuvent ne pas être exactes.

Figure 2: Use of the `%ai_question` command.

Should the user prefer to use multiple lines to phrase the question (e.g. give a snippet of code and ask questions about it), it is possible to use the adapted command `%%ai_question`, with a double `%`, at the beginning of an empty code cell, followed by the question using multiple lines, and run it. The screenshot in [Figure 3](#) illustrates the use of the `%%ai_question` command with multiple lines.

```
[18]: %%ai_question Pour quoi on a besoin de vérifier si matrix.shape[0] == matrix.shape[1] dans cette fonction :
def calculate_determinant(matrix):
    if matrix.shape[0] == matrix.shape[1]:
        determinant = np.linalg.det(matrix)
        return determinant
    return None
```

Excellente question !

Dans le code que vous avez fourni, la vérification `if matrix.shape[0] == matrix.shape[1]:` est nécessaire pour s'assurer que la matrice est carrée, c'est-à-dire que le nombre de lignes est égal au nombre de colonnes. Pourquoi est-ce important ?

En effet, le déterminant d'une matrice ne peut être calculé que pour les matrices carrées. Si la matrice n'est pas carrée, le déterminant n'est pas défini. En d'autres termes, la fonction `np.linalg.det()` attend une matrice carrée comme entrée pour calculer le déterminant.

En vérifiant que `matrix.shape[0]` (le nombre de lignes) est égal à `matrix.shape[1]` (le nombre de colonnes), on s'assure que la matrice est carrée et qu'il est donc possible de calculer le déterminant. Si la matrice n'est pas carrée, la fonction retourne `None`, indiquant que le déterminant ne peut pas être calculé.

En rappel, nous avons vu les propriétés des matrices lors de notre cours de mathématiques pour la science des données, notamment que les opérations sur les matrices sont souvent définies uniquement pour les matrices carrées. C'est pourquoi cette vérification est cruciale pour éviter une erreur lors du calcul du déterminant.

En résumé, cette vérification est une précaution pour s'assurer que le déterminant peut être calculé sans erreur, car il n'est pas défini pour les matrices non carrées.

⚠️ Réponse générée par une intelligence artificielle, les informations peuvent ne pas être exactes.

Figure 3: Use of the `%%ai_question` command for questions with multiple lines.

4 Using the `%ai_code` command

The `%ai_code` command is designed for code generation. To use the `%ai_code` command, the user simply needs to place “`%ai_code`” at the beginning of an empty code cell, followed by the instructions for the code generation on the same line, and run it. The toolkit will display the generation in real time (token by token) as a markdown, relocating only the python code to a new cell directly below the instructions once the generation is complete. The screenshot in [Figure 4](#) illustrates the use of the command.

```
[6]: %ai_code Fonction qui calcule le déterminant d'une matrice numpy
```

This code defines a function `calculate_determinant` that calculates the determinant of a given numpy matrix. It first checks if the input matrix is square (i.e., has the same number of rows and columns) and raises a `ValueError` if it's not. Then, it uses the `np.linalg.det` function from the numpy library to calculate the determinant. The example usage shows how to use this function with a valid square matrix and an invalid non-square matrix, demonstrating the error handling.

Le code généré par l'IA a été inséré dans la cellule ci-dessous

```
[ ]: # ⚠ Code généré par une intelligence artificielle sujet à des erreurs
# Instructions pour la génération : 'Fonction qui calcule le déterminant d'une matrice numpy'

import numpy as np
def calculate_determinant(matrix):
    """
    Calculate the determinant of a numpy matrix.
    Args:
        matrix (numpy array): A 2D numpy array representing a square matrix.
    Returns:
        float: The determinant of the matrix.
    Raises:
        ValueError: If the input matrix is not a square matrix.
    """
    # Check if the matrix is square
    if matrix.shape[0] != matrix.shape[1]:
        raise ValueError("The input matrix must be a square matrix.")

    # Calculate the determinant using numpy's linalg.det function
    determinant = np.linalg.det(matrix)
    return determinant


# Example usage:
matrix = np.array([[1, 2], [3, 4]])
print("Determinant of the matrix:", calculate_determinant(matrix)) # Output: Determinant of the matrix: -2.0
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
try:
    calculate_determinant(matrix)
except ValueError as e:
    print(e) # Output: The input matrix must be a square matrix.
```

Figure 4: Use of the `%ai_code` command.

Should the user prefer to use multiple lines to phrase the instructions (e.g. give a snippet of code and ask the model to re-write an optimized version), it is possible to use the adapted command `%%ai_code`, with a double `%`, at the beginning of an empty code cell, followed by the instructions using multiple lines, and run it. The screenshot in [Figure 5](#) illustrates the use of the `%%ai_code` command with multiple lines.

```
[20]: %%ai_code Je veux un code qui fasse la même chose que celui-ci mais de manière plus élégante :
def is_larger(a, b):
    if (a > b):
        return True
    elif(a == b):
        return False
    elif(a < b):
        return False
```

This code does the same thing as your original function, but in a more elegant and concise way. The `return a > b` statement directly returns the result of the comparison, which is a boolean value (True or False), making the if-elif statements unnecessary. This code is more readable and easier to understand.

Le code généré par l'IA a été inséré dans la cellule ci-dessous 

```
[ ]: # ⚠ Code généré par une intelligence artificielle sujet à des erreurs
...
Instructions pour la génération :
Je veux un code qui fasse la même chose que celui-ci mais de manière plus élégante :
def is_larger(a, b):
    if (a > b):
        return True
    elif(a == b):
        return False
    elif(a < b):
        return False
...

def is_larger(a, b):
    """
    Return True if a is larger than b, False otherwise.
    Parameters:
        a (int or float): the first number
        b (int or float): the second number
    Returns:
        bool: True if a is larger than b, False otherwise
    """
    return a > b

# Example usage:
print(is_larger(5, 3)) # Output: True
print(is_larger(3, 5)) # Output: False
print(is_larger(5, 5)) # Output: False
```

Figure 5: Use of the `%%ai_code` command for instructions with multiple lines.

5 Using the `%%ai_explain` command

The `%%ai_explain` command is designed for code explanation. To use the `%%ai_explain` command, the user has two options:

- Place the command on the first line of a new cell, followed on the same line by the number of the cell containing the code to be explained (it must have been executed in the current session);
- Place the adapted command `%%ai_explain`, with a double %, on the first line of the cell containing the code to be explained. The actual code of the cell will not be executed; only its explanation will be generated.

```
[15]: import numpy as np
def calculate_determinant(matrix):
    if matrix.shape[0] != matrix.shape[1]:
        raise ValueError("The input matrix must be a square matrix.")
    determinant = np.linalg.det(matrix)
    return determinant

matrix = np.array([[1, 2], [3, 4]])
print("Determinant of the matrix:", calculate_determinant(matrix)) # Output: Determinant of the matrix: -2.0
Determinant of the matrix: -2.0000000000000004
```

```
[16]: %ai_explain 15
```

Bonjour ! Dans ce code Python, nous allons analyser une fonction qui calcule le déterminant d'une matrice carrée (ou square matrix) à l'aide de la bibliothèque NumPy.

L'objectif global de ce code est de créer une fonction `calculate_determinant` qui prend en entrée une matrice carrée et qui retourne son déterminant. Pour cela, le programmeur utilise la bibliothèque NumPy, qui est une bibliothèque de calcul numérique en Python, très utile pour les opérations matricielles. La fonction commence par vérifier si la matrice est carrée, c'est-à-dire si le nombre de lignes est égal au nombre de colonnes. Si ce n'est pas le cas, elle lance une erreur `ValueError` avec un message indiquant que la matrice doit être carrée. Si la matrice est carrée, elle utilise la fonction `np.linalg.det` de NumPy pour calculer le déterminant de la matrice.

```
if matrix.shape[0] != matrix.shape[1]:
    raise ValueError("The input matrix must be a square matrix.")
```

La logique utilisée est la suivante : la fonction vérifie d'abord si la matrice est carrée en comparant le nombre de lignes (`matrix.shape[0]`) et le nombre de colonnes (`matrix.shape[1]`). Si ces deux valeurs sont différentes, elle lance une erreur. Sinon, elle utilise la fonction `np.linalg.det` pour calculer le déterminant de la matrice.

Les principales variables utilisées sont :

- `matrix` : la matrice en entrée de la fonction `calculate_determinant`.
- `determinant` : le résultat du calcul du déterminant de la matrice.

La bibliothèque utilisée est :

- `numpy` (importée sous le nom `np`) : une bibliothèque de calcul numérique en Python qui fournit des outils pour les opérations matricielles et des fonctions pré-construites pour les calculs numériques.

La fonction utilisée est :

- `np.linalg.det` : une fonction de NumPy qui calcule le déterminant d'une matrice carrée.

En fin de code, le programmeur crée une matrice `matrix` avec deux lignes et deux colonnes, puis appelle la fonction `calculate_determinant` avec cette matrice en argument, et affiche le résultat.

```
matrix = np.array([[1, 2], [3, 4]])
print("Determinant of the matrix:", calculate_determinant(matrix)) # Output: Determinant of the matrix: -2.0
```

Ce code pourrait être utilisé dans des applications qui nécessitent des calculs matriciels, comme la résolution de systèmes d'équations linéaires, la transformation de coordonnées ou la modélisation de phénomènes physiques.

En résumé, ce code définit une fonction `calculate_determinant` qui prend une matrice carrée en entrée et retourne son déterminant, en utilisant la bibliothèque NumPy pour effectuer ce calcul. La fonction vérifie d'abord si la matrice est carrée, puis utilise la fonction `np.linalg.det` pour calculer le déterminant. Ce code est utile pour les calculs matriciels et les applications qui en dépendent.

Figure 6: Use of the `%ai_explain` command on a new cell.

Figure 6 illustrates the use of the command in a new cell, while Figure 7 illustrates the use of the adapted command `%%ai_explain` in the original cell with the code to be explained. Note that, as the message displayed before the explanation shows, it is necessary to remove the command from the first line in order to execute the code itself inside the cell instead of generating an explanation of it.


```
[17]: %%ai_explain
import numpy as np
def calculate_determinant(matrix):
    if matrix.shape[0] != matrix.shape[1]:
        raise ValueError("The input matrix must be a square matrix.")
    determinant = np.linalg.det(matrix)
    return determinant

matrix = np.array([[1, 2], [3, 4]])
print("Determinant of the matrix:", calculate_determinant(matrix)) # Output: Determinant of the matrix: -2.0
```

⚠ Note : Afin d'exécuter le contenu de cette cellule plutôt que d'en générer une explication, vous devez supprimer la commande magique `%%ai_explain` sur la première ligne et exécuter cette cellule à nouveau.

Bonjour ! Aujourd'hui, nous allons analyser un code Python qui calcule le déterminant d'une matrice carrée.

L'objectif de ce code est de créer une fonction qui prend en entrée une matrice carrée et retourne son déterminant. Pour cela, le programmeur utilise la bibliothèque NumPy, qui est une bibliothèque de traitement de données et d'algèbre linéaire très puissante en Python. La fonction `calculate_determinant` prend en entrée une matrice et vérifie d'abord si elle est carrée, c'est-à-dire si elle a le même nombre de lignes et de colonnes. Si ce n'est pas le cas, la fonction lance une erreur (`ValueError`) car le déterminant n'est défini que pour les matrices carrées. Si la matrice est carrée, la fonction utilise la fonction `np.linalg.det` de NumPy pour calculer le déterminant de la matrice.

```
if matrix.shape[0] != matrix.shape[1]:
    raise ValueError("The input matrix must be a square matrix.")
```

Cette partie du code vérifie que la matrice a la même taille pour les lignes et les colonnes en comparant les éléments `shape[0]` et `shape[1]` de la matrice, qui représentent respectivement le nombre de lignes et de colonnes. Si ces valeurs sont différentes, la fonction lève une erreur.

Ensuite, la fonction utilise la fonction `np.linalg.det` pour calculer le déterminant de la matrice. Cette fonction est très rapide et efficace car elle utilise des algorithmes optimisés pour le calcul du déterminant.

```
determinant = np.linalg.det(matrix)
```

Enfin, la fonction retourne le déterminant calculé.

Les principales variables utilisées dans ce code sont : • `matrix` : la matrice carrée en entrée de la fonction `calculate_determinant` • `determinant` : le résultat du calcul du déterminant de la matrice

Ensuite, le code crée une matrice 2x2 avec les éléments `[[1, 2], [3, 4]]` et l'assigne à la variable `matrix`. Enfin, il appelle la fonction `calculate_determinant` avec cette matrice en argument et affiche le résultat.

Ce code est très utile pour les étudiants en mathématiques et en physique qui doivent souvent calculer le déterminant de matrices pour résoudre des systèmes d'équations linéaires ou diagonaliser des matrices. Par exemple, si vous devez résoudre un système d'équations linéaires représenté par une matrice, vous pouvez utiliser ce code pour vérifier si la matrice est inversible (c'est-à-dire si son déterminant est non nul).

En résumé, ce code définit une fonction `calculate_determinant` qui prend une matrice carrée en entrée et retourne son déterminant en utilisant la bibliothèque NumPy. Il vérifie d'abord si la matrice est carrée, puis utilise la fonction `np.linalg.det` pour calculer le déterminant. Enfin, il crée une matrice exemple et affiche son déterminant.

Figure 7: Use of the `%%ai_explain` command on the original cell.

6 Using the `%%ai_debug` command

The `%%ai_debug` command is designed for code debugging (i.e. codes that are displaying an error message when executed). To use the `%%ai_debug` command, the user has two options:

- Place the command on the first line of a new cell, followed on the same line by the number of the cell containing the code to be debugged (it must have been executed in the current session);
- Place the adapted command `%%ai_debug`, with a double %, on the first line of the cell containing the code to be debugged. The actual code of the cell will not be executed; only its debugging analysis will be generated.

Note that this command is designed only for codes that display an error message when executed. For codes that run but generate unexpected results, you can use the `%%ai_question` command to ask specific questions.


```
[10]: def add_one(my_list):
      len_list = len(my_list)
      start = 1
      end = start + len_list
      for i in range(start, end):
          my_list[i] += 1
      return my_list

      example_list = [12, 4, 88, 64, 2, 79, 108]
      new_list = add_one(example_list)
```

```
-----
IndexError                                Traceback (most recent call last)
Cell In[10], line 10
      7     return my_list
      9     example_list = [12, 4, 88, 64, 2, 79, 108]
----> 10 new_list = add_one(example_list)

Cell In[10], line 6, in add_one(my_list)
      4 end = start + len_list
      5 for i in range(start, end):
----> 6     my_list[i] += 1
      7 return my_list

IndexError: list index out of range
```

```
[11]: %ai_debug 10
```

Rapport de débogage:

Le code ci-dessus définit une fonction `add_one` qui prend une liste en entrée, mais elle rencontre une erreur `list index out of range` lors de l'exécution. L'analyse de cette erreur révèle que le problème vient de la façon dont la boucle `for` est utilisée pour mettre à jour les éléments de la liste.

La variable `end` est définie comme `start + len_list`, ce qui signifie que la boucle `for` essaiera d'accéder à des indices qui sont en dehors de la plage de la liste, car les indices de liste en Python commencent à 0 et vont jusqu'à `len_list - 1`. Par exemple, si la liste a une longueur de 6, la boucle essaiera d'accéder aux indices 1 à 7, ce qui est hors de la plage de la liste.

De plus, la boucle `for` essaie de modifier les éléments de la liste en utilisant l'opérateur `+=`, mais comme les indices sont hors de la plage de la liste, cela génère l'erreur `list index out of range`.

Solution:

```
def add_one(my_list):
    for i in range(len(my_list)):
        my_list[i] += 1
    return my_list

example_list = [12, 4, 88, 64, 2, 79, 108]
new_list = add_one(example_list)
```

Explications:

- La boucle `for` est modifiée pour utiliser `range(len(my_list))`, qui génère des indices allant de 0 à `len_list - 1`, ce qui correspond à la plage d'indices valides pour la liste.
- La variable `start` est supprimée, car elle n'est pas nécessaire dans ce contexte.
- La variable `end` est également supprimée, car elle n'est plus utile.
- La boucle `for` met à jour les éléments de la liste en utilisant l'indice `i` comme index, ce qui est maintenant valide.

Figure 8: Use of the `%ai_debug` command on a new cell.

Figure 8 illustrates the use of the command in a new cell, while Figure 9 illustrates the use of the adapted command `%%ai_debug` in the original cell with the code to be debugged. Note that, as the message displayed before the explanation shows, it is necessary to remove the command from the first line in order to execute the code itself inside the cell instead of generating a debugging analysis of it.

```
[9]: %%ai_debug
def add_one(my_list):
    len_list = len(my_list)
    start = 1
    end = start + len_list
    for i in range(start, end):
        my_list[i] += 1
    return my_list

example_list = [12, 4, 88, 64, 2, 79, 108]
new_list = add_one(example_list)
```

⚠ Note : Après avoir corrigé le bug, afin de bien exécuter le contenu de cette cellule plutôt que d'en générer un rapport de débogage, vous devez supprimer la commande magique `%%ai_debug` sur la première ligne.

Rapport de débogage:

Le code ci-dessus essaie de définir une fonction `add_one` qui prend une liste en entrée et ajoute 1 à chaque élément de la liste. Cependant, il rencontre une erreur `list index out of range` lors de l'exécution.

Analyse de l'erreur: L'erreur `list index out of range` signifie que le code est en train d'essayer d'accéder à un indice de liste qui est en dehors de la plage de la liste. Dans ce cas, la erreur se produit dans la boucle `for` où le code essaie d'accéder à `my_list[i]` avec `i` allant de `start` (1) à `end` (taille de la liste + 1). Le problème est que les indices de liste en Python commencent à 0, donc si la liste a une taille de 6, les indices valides sont de 0 à 5. En démarrant la boucle à 1 et en allant jusqu'à `len_list + 1`, le code dépasse la taille de la liste, ce qui cause l'erreur.

Correction proposée:

```
def add_one(my_list):
    for i in range(len(my_list)): # Corrige la plage d'indices pour démarrer à 0
        my_list[i] += 1
    return my_list

example_list = [12, 4, 88, 64, 2, 79, 108]
new_list = add_one(example_list)
```

Explication: La correction modifie la plage d'indices de la boucle `for` pour démarrer à 0 et aller jusqu'à la taille de la liste, évitant ainsi l'erreur `list index out of range`. Maintenant, la fonction `add_one` ajoutera 1 à chaque élément de la liste sans erreur.

Note: Si vous voulez créer une nouvelle liste avec les éléments augmentés de 1, vous pouvez utiliser une liste comprehension :

```
def add_one(my_list):
    return [x + 1 for x in my_list]

example_list = [12, 4, 88, 64, 2, 79, 108]
new_list = add_one(example_list)
```

Figure 9: Use of the `%%ai_debug` command on the original cell.

7 Using the `%%ai_quiz` command

The `%%ai_quiz` command is designed for quiz generation. To use the `%%ai_quiz` command, the user has three options:

- Place the command by itself on the first line of an empty cell: to generate a quiz on a random lesson (course notebook);
- Place the command on the first line of a new cell, followed by the desired quiz subject on the same line: to generate a quiz on a specific subject;
- Place the command on the first line of a new cell, followed by the name of a lesson (course notebook) in CMx or CMx.md format (e.g. CM2 or CM2.md) on the same line: to generate a quiz on a specific lesson.

```
[4]: %ai_quiz réseau de neurones
```

Question 1/5: Quel est le rôle de la fonction d'activation dans un réseau de neurones ?

Choisissez l'option correcte:

- ☐ Augmenter la complexité du modèle
- ☐ Améliorer la précision du modèle
- ☐ Réduire les coûts de calcul
- ☐ Introduire de la non-linéarité dans les calculs

Précédent

Suivant

Figure 10: Use of the `%ai_quiz` command.

The screenshot in **Figure 10** illustrates the use of the command, while **Figure 11** shows the result page once the quiz is completed. Then, the user can click on the Feedback button to get a full LLM-generated review, discussing the questions and explaining the answers. The generation of the feedback will be displayed in real time (token by token) as a markdown.

```
[4]: %ai_quiz réseau de neurones
```

Quiz terminé !

✓ Question 1: Quel est le rôle de la fonction d'activation dans un réseau de neurones ?

Réponse Sélectionnée : Introduire de la non-linéarité dans les calculs

✓ Question 2: Quel est le principe de base de l'algorithme de rétropropagation du gradient ?

Réponse Sélectionnée : Minimiser la fonction de perte en ajustant les poids

✓ Question 3: Qu'est-ce que la sur-ajustement dans un réseau de neurones ?

Réponse Sélectionnée : Un phénomène où le modèle s'adapte trop bien aux données d'apprentissage

✗ Question 4: Quel est le but de la normalisation des données en entrée d'un réseau de neurones ?

Réponse Sélectionnée : Réduire le nombre de neurones

Réponse Correcte : Éviter les problèmes de scalabilité

✓ Question 5: Quel est le rôle de la couche d'output dans un réseau de neurones ?

Réponse Sélectionnée : Produire la sortie finale du modèle

📋 Votre nombre de réponses correctes est de 4/5.

Feedback

Figure 11: Result page of the `%ai_quiz` command after completing the quiz.