Natural Language Processing **Tel Aviv University**

Assignment 1: Word Representations

Due Date: *November 30 , 2025* Lecturer: Dr. Tal Wagner, TA: Idan Tarshish

Preliminaries

Submission Instructions This assignment includes both theoretical and coding questions. Your submission should include:

1. A single PDF file in digital format (we recommend using Overleaf as in HW0, but you may use any digital editor you prefer). The PDF should contain:
   (a) `word_vectors.png` from Q2e, and your answers to Q2f.
   (b) Your answers to Q3 and Q4.
   (c) For Q5e — the generated samples and your explanations.
   (d) Your answers to Q6.
   (e) The AI Disclosure & Reflection (Q7).
2. The directory `q2_skeleton`, including all its files, with the **completed** `.py` files.
3. The **completed** notebook `q5_DAN.ipynb`.
4. The **completed** notebook `q6_topic_modeling.ipynb`.

Submit your solution via Moodle. Your submission should be a single zip file named `<id1>_<id2>_<id3>.zip`, where `id1` is the ID of the first student. The zip file should include all of the above components.

Notes:

- Only one student needs to submit.
- Your implementation should be efficient and vectorized whenever possible (i.e., use numpy matrix operations rather than `for` loops). A non-vectorized implementation will not receive full credit!
- Your code should follow coding standards (well-documented, self-explained, meaningful naming, PEP8).

Acknowledgements This assignment was adapted from Stanford's CS224n course. Their contributions are greatly appreciated. In this note, we also like to acknowledge the work of the previous TAs of this course for building this exercise.

1 Understanding word2vec (theoretical)

Let's have a quick refresher on the `word2vec` algorithm.

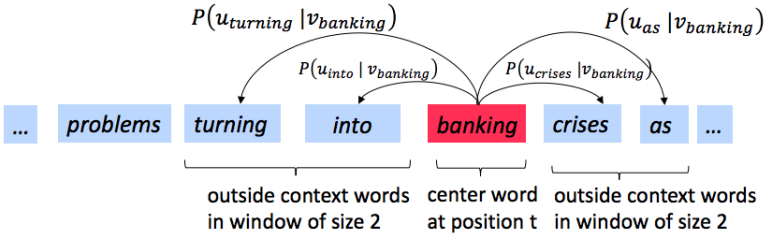

Figure 1: The word2vec skip-gram prediction model with window size 2

There is no need to submit this part. It's meant for you to verify your understanding (if you are not sure why learning it is significant here is some *motivation*). You will need close form of the partial derivative to implement the following section (you are not allowed to use automatic differentiation packages here). For further reading: *Stanford cs224n gradient notes*.

The key insight behind `word2vec` is that '*a word is known by the company it keeps*'. Concretely, suppose we have a 'center' word c and a contextual window surrounding c. We shall refer to words that lie in this contextual window as 'outside words'. For example, in Figure 1 we see that the center word c is 'banking'. Since the context window size is 2, the outside words are 'turning', 'into', 'crises', and 'as'.

The goal of the skip-gram `word2vec` algorithm is to accurately learn the probability distribution $P(O \mid C)$. Given a specific word o and a specific word c, we want to calculate $P(O = o \mid C = c)$, which is the probability that word o is an 'outside' word for c, i.e., the probability that o falls within the contextual window of c.

In word2vec, the conditional probability distribution is given by taking vector dot-products and applying the softmax function:

$$P(O = o \mid C = c) = \frac{\exp\left(\boldsymbol{u}_o^\intercal \boldsymbol{v}_c\right)}{\sum_{w \in \mathrm{W}} \exp\left(\boldsymbol{u}_w^\intercal \boldsymbol{v}_c\right)} \tag{1}$$

Here, W is the vocabulary, $\boldsymbol{u}_o$ is the 'outside' vector representing outside word o, and $\boldsymbol{v}_c$ is the 'center' vector representing center word c. To contain these parameters, we have two matrices, $\boldsymbol{U}$ and $\boldsymbol{V}$. The rows of $\boldsymbol{U}$ are all the 'outside' vectors $\boldsymbol{u}_w$. The rows of $\boldsymbol{V}$ are all of the 'center' vectors $\boldsymbol{v}_w$. Both $\boldsymbol{U}$ and $\boldsymbol{V}$ contain a vector for every $w \in W$[1].

Recall from lectures that, for a single pair of words c and o, the loss is given by:

$$\boldsymbol{J}_{\text{naïve-softmax}}(c, o, \boldsymbol{V}, \boldsymbol{U}) = -\log P(O = o \mid C = c) \tag{2}$$

Another way to view this loss is as the cross-entropy[2] between the true distribution $\boldsymbol{y}$ and the predicted distribution $\hat{\boldsymbol{y}}$. Here, both $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are vectors with length equal to the number of words in the vocabulary ($|W|$). Furthermore, the k^{th} entry in these vectors indicates the conditional probability of the k^{th} word being an 'outside word' for the given c. The true empirical distribution $\boldsymbol{y}$ is a one-hot vector with a 1 for the true outside word o, and 0 everywhere else. The predicted distribution $\hat{\boldsymbol{y}}$ is the probability distribution $P(O \mid C = c)$ given by our model in Equation (1).

In this section we're going to view many different derivatives. These derivatives will be used in the next

[1] Assume that every word in our vocabulary is matched to an integer number k. $\boldsymbol{u}_k$ is both the k^{th} row of $\boldsymbol{U}$ and the 'outside' word vector for the word indexed by k. $\boldsymbol{v}_k$ is both the k^{th} row of $\boldsymbol{V}$ and the 'center' word vector for the word indexed by k. **In order to simplify notation we shall interchangeably use k to refer to the word and the index-of-the-word.**

[2] The Cross Entropy Loss between the true (discrete) probability distribution p and another distribution q is $-\sum_i p_i \log(q_i)$.

section, where you'll be tasked to implement this algorithm (so you will need to have a closed-form of the derivative, as we're not using automatic differentiation packages here).

(a) Equation (1) uses the softmax function. Recall you should in HW0 that softmax is invariant to constant offset in the input, i.e that for any input vector $\boldsymbol{x}$ and any constant c,

$$\text{softmax}(\boldsymbol{x}) = \text{softmax}(\boldsymbol{x} + c)$$

where $\boldsymbol{x} + c$ means adding the constant c to every dimension of $\boldsymbol{x}$. Remember that

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

(b) Verify you understand why the naïve softmax loss given in Equation (2) is the same as the cross-entropy loss between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$, i.e., show that

$$-\sum_{w \in \text{W}} y_w \log(\hat{y}_w) = -\log(\hat{y}_o) \tag{3}$$

(c) Recall that in class we showed that the partial derivative of $\boldsymbol{J}_{\text{naïve-softmax}}(c, o, \boldsymbol{V}, \boldsymbol{U})$ with respect to $\boldsymbol{v}_c$ is $\boldsymbol{u}_o - \mathbb{E}_{o' \sim p(o'|c)}[\boldsymbol{u}_{o'}]$. Now, Compute the partial derivative of $\boldsymbol{J}_{\text{naïve-softmax}}(c, o, \boldsymbol{V}, \boldsymbol{U})$ with respect to each of the 'outside' word vectors, $\boldsymbol{u}_w$'s. There will be two cases: when $w = o$, the true 'outside' word vector, and when $w \neq o$, for all other words.

You should verify you understand why the answer is as follow:

$$\frac{\partial}{\partial u_{w \neq o}} \mathbf{J}_{naive-softmax}(c, o, \mathbf{V}, \mathbf{U}) = \hat{y_w} v_c$$

$$\frac{\partial}{\partial u_o} \mathbf{J}_{naive-softmax}(c, o, \mathbf{V}, \mathbf{U}) = v_c(y^T \hat{y} - 1)$$

Now we shall consider the negative sampling loss, which is an alternative to the naïve softmax loss. Assume that K negative samples (words) are drawn from the vocabulary W. For simplicity of notation we shall refer to them as $w_1, w_2, \ldots, w_K$ and to their corresponding outside vectors as $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_K$. Note that $o \notin \{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_K\}$. For a center word c and an outside word o, the negative sampling loss function is given by:

$$\boldsymbol{J}_{\text{neg-sample}}(c, o, \boldsymbol{V}, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\intercal \boldsymbol{v}_c))) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\intercal \boldsymbol{v}_c))$$

for a sample $w_1, w_2, \ldots, w_K$, where $\sigma(\cdot)$ is the sigmoid function[3].

The partial derivative of this loss with respect to $\boldsymbol{v}_c$ is: $-(1 - \sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c))u_o + \sum_{k=1}^{K} (1 - \sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c))\boldsymbol{u}_k$.

Now, with respect to $\boldsymbol{u}_o$: $-(1 - \sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c))v_c$.

And with respect to $\boldsymbol{u}_k$: $(1 - \sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c))v_c$.

You'll use these derivatives while implementing the algorithm in next section.

[3]The loss function here is the negative of what Mikolov et al. had in their original paper, because we are minimizing rather than maximizing in our assignment code. Ultimately, this is the same objective function.

(d) Suppose the center word is $c = w_t$ and the context window is $[w_{t-m}, \ldots, w_{t-1}, w_t, w_{t+1}, \ldots, w_{t+m}]$, where m is the context window size. Recall that for the skip-gram version of `word2vec`, the total loss for the context window is:

$$\boldsymbol{J}_{\text{skip-gram}}(c, w_{t-m}, \ldots, w_{t+m}, \boldsymbol{V}, \boldsymbol{U}) = \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \boldsymbol{J}(c, w_{t+j}, \boldsymbol{V}, \boldsymbol{U})$$

Here, $\boldsymbol{J}(c, w_{t+j}, \boldsymbol{V}, \boldsymbol{U})$ represents an arbitrary loss term for the center word $c = w_t$ and outside word w_{t+j}. $\boldsymbol{J}(c, w_{t+j}, \boldsymbol{V}, \boldsymbol{U})$ could be $\boldsymbol{J}_{\text{naïve-softmax}}(c, w_{t+j}, \boldsymbol{V}, \boldsymbol{U})$ or $\boldsymbol{J}_{\text{neg-sample}}(c, w_{t+j}, \boldsymbol{V}, \boldsymbol{U})$, depending on your implementation.

Write down three partial derivatives:

(i) $\partial \boldsymbol{J}_{\text{skip-gram}}(c, w_{t-m}, \ldots, w_{t+m}, \boldsymbol{V}, \boldsymbol{U}) / \partial \boldsymbol{U}$

(ii) $\partial \boldsymbol{J}_{\text{skip-gram}}(c, w_{t-m}, \ldots, w_{t+m}, \boldsymbol{V}, \boldsymbol{U}) / \partial \boldsymbol{v}_c$

(iii) $\partial \boldsymbol{J}_{\text{skip-gram}}(c, w_{t-m}, \ldots, w_{t+m}, \boldsymbol{V}, \boldsymbol{U}) / \partial \boldsymbol{v}_w$ when $w \neq c$

Write your answers in terms of $\partial \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}) / \partial \boldsymbol{U}$ and $\partial \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}) / \partial \boldsymbol{v}_c$.

You should verify you understand why the answer is as follow:

(i)

$$\sum_{-m \leq j \leq m} \frac{\partial \mathbf{J}(v_c, w_{t+j}, V, U)}{\partial U}$$

(ii)

$$\sum_{-m \leq j \leq m} \frac{\partial \mathbf{J}(v_c, w_{t+j}, V, U)}{\partial v_c}$$

(iii)

$$0$$

Once you're done: *Given that you computed the derivatives of $\boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$ with respect to all the model parameters $\boldsymbol{U}$ and $\boldsymbol{V}$ in parts (a) to (c), you have now computed the derivatives of the full loss function $\boldsymbol{J}_{skip\text{-}gram}$ with respect to all parameters. You're ready to implement* `word2vec`*!*

(e) Try to explain in a few sentences why it is important to split each token representation into two - first for its being the center token, and second for it being an output token.

(f) Finally, try to explain the intuition behind this algorithm. That is, why we might expect this algorithm to lead the model end up representing two semantically similar tokens with two "close" vectors within the Euclidean space.

2 Implementing word2vec (code implementation) (25 points)

In this part you will implement the word2vec model (using the knowledge you gained in the previous section) and train your own word vectors with stochastic gradient descent (SGD).

A standard Python environment should be sufficient for this exercise, requiring `Python` $\geq$ `3.5.x`, along with the `numpy` and `matplotlib` packages. If you encounter any issues installing or using these packages, please refer to the "Setup Virtual Environment with Conda" instructions at the end of this section, which explain how to create a new virtual environment.

This section relies heavily on the previous theoretical section. If any of the functions you are required to implement are unclear, please refer back to the previous section for clarification.

(a) Implement the `softmax` function in the module `q2a_softmax.py`. Note that in practice, for numerical stability, we make use of the property we proved in question 1.a and choose $c = -\max_i x_i$ when computing softmax probabilities (i.e., subtracting its maximum element from all elements of x). You can test your implementation by running `python q2a_softmax.py`.

(b) To make debugging easier, we will now implement a gradient checker. This function compares the analytical gradients returned by a function to their numerical approximations. Follow the instructions provided inside the module, and complete the implementation of the `gradcheck_naïve` function in the file `q2b_gradcheck.py`. You can test your implementation by running `python q2b_gradcheck.py`.

(c) Fill in the implementation for `naïve_softmax_loss_and_gradient`, `neg_sampling_loss_and_gradient`, and `skipgram` in the module `q2c_word2vec.py`. You can test your implementation by running `python q2c_word2vec.py`. Verify that your results are approximately equal to the expected results.

(d) Complete the implementation for the SGD optimizer in the module `q2d_sgd.py`. You can test your implementation by running `python q2d_sgd.py`.

(e) Show time! Now we are going to load some real data and train word vectors with everything you just implemented! We are going to use the Stanford Sentiment Treebank (SST) dataset to train word vectors, and later apply them to a simple sentiment analysis task. There is no additional code to write for this part; just run `python q2e_run.py`.

Note: The training process may take a long time depending on the efficiency of your implementation. Plan accordingly!

After 40,000 iterations, the script will finish and a visualization for your word vectors will appear. It will also be saved as `word_vectors.png` in your project directory. **Include the plot in your homework write up, inside the pdf (not a separate file)**.

(f) Include in your PDF brief answers to the following questions (in no more than 10 lines of text):

1) Explain what you notice in the plot. Are there any reasonable clusters/trends?

2) Are the word vectors as good as you expected? What do you think could make them better?

2.1 Setup Virtual Environment with Conda

To create the venv, first run the following commands within the assignment directory. This guarantees that you have all the necessary packages to complete the assignment.

```
conda env create --file env.yml
conda activate nlp-hw1
```

Once you are done with the assignment you can deactivate this environment by running:

```
conda deactivate
```

Note: you are not required to use conda in this assignment, but the usage of virtual environment managers (such as conda) is highly recommended and is standard practice in both the industry and the academia. If you are unfamiliar with conda, check out `https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html`.

3 Optimizing word2vec (theoretical) (15 points)

We will now prove that in the skipgram algorithm, the maximum likelihood solution for word embedding probabilities is their empirical distribution, and show that there exists a scenario where reaching the optimum is impossible.

(a) For a corpus of length T, recall that the objective is:

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} p_\theta \left(w_{t+j} \mid w_t \right)$$
$$J(\theta) = \log \mathcal{L}(\theta) = \sum_{t=1}^{T} \sum_{-m \leq j \leq m} \log p_\theta \left(w_{t+j} \mid w_t \right)$$

Prove that if $\theta^* = \arg\max_\theta \mathcal{L}(\theta)$ then $p_{\theta^*}(o \mid c) = \frac{\#(c,o)}{\sum_{o'} \#(c,o')}$.
Where $\#(c, o) =$ Number of co-occurrence of c and o in the corpus.

Hint 1: For a fixed c, o in the vocabulary, how many times does the term $p_\theta(o \mid c)$ appear in $\mathcal{L}(\theta)$
Hint 2: Use Lagrange multipliers.

(b) Recall that in the Skip-Gram model we use a softmax function to model the probability of a context word b given a center word a:

$$p(b|a) = \frac{e^{u_b v_a}}{\sum_o e^{u_o v_a}}$$

where v_a denotes the embedding of the center (input) word, and u_b denotes the embedding of the context (output) word. Let's assume each word is represented by a single scalar (real number). Prove that there is a corpus over a vocabulary of no more than 4 words, where reaching the optimum solution is impossible. You can assume that a corpus is a list of sentences, such as {"*aa*", "*bb*", "*cc*", . . . , } - each letter represnet a word.

For instance, for the given corpus: $\{aa, aa, aa, ab, ab, ac\}$ the optimal solution will be:

$$p(a|a) = \frac{6}{9} = \frac{2}{3}$$
$$p(b|a) = \frac{2}{9}$$
$$p(c|a) = \frac{1}{9}$$

4 Paraphrase Detection (theoretical) (15 points)

Paraphrase detection is a binary classification task, where given two sentences, the model needs to determine if they are paraphrases of one another. For example the pair of sentences *"The school said that the buses accommodate 24 students"* and *"It was said by the school that the buses seat two dozen students"* are paraphrases. While the pair *"John ate bread"* and *"John drank juice"* are not.

Let's denote by x_1, x_2 the input pair of sentences and by $\mathbf{x}_1, \mathbf{x}_2$, a vector representation for the pair of sentences obtained using some neural network, where $\mathbf{x}_i \in \mathbb{R}^d$.

Consider the following model for paraphrase detection:

$$p(\text{the pair is a paraphrase} \mid x_1, x_2) = \sigma\left(\text{relu}(\mathbf{x}_1)^\top \text{relu}(\mathbf{x}_2)\right),$$

where $\text{relu}(x) = \max(0, x)$, σ is sigmoid function. The model output is True if $p(\text{the pair is a paraphrase} \mid x_1, x_2) \geq 0.5$ and False otherwise.

(a) In this model, what is the maximal accuracy on a dataset where the ratio of positive to negative examples is 1:4?

(b) Suggest a simple fix for the problem.

(c) What evaluation metrics should you use to evaluate the success of a model on this imbalanced dataset? Choose 2-3 metrics, and explain your choice in 10 sentences (or less). Consider: Accuracy, precision, Recall, ROC-AUC, AUC-PR, confusion matrix.

5 Deep Averaging Networks (code implementation) (25 points)

In this question we will implement Deep Averaging Networks (DAN), and work on the IMDB dataset. The IMDB dataset consists of positive and negative reviews for movies.
This exercise will also introduce you to the popular `transformers` package from Hugging Face. It will also require reading some of the documentation of `pytorch`. The total number of lines of code you need to write for implementing the model itself is roughly 10 or less (i.e., not a lot). Complete the code in the attached notebook, and answer the following questions:

Note: The model in the paper uses GloVe embeddings, in this exercise, you will implement this model using GloVe embedding trained on slightly less data, so you should expect different results then the ones shown in the paper.

Note 2: Make sure to show the code you run for each sub-question. Whenever possible, avoid modifying or overwriting earlier code in the notebook. You may, however, adjust the DAN class init parameters as needed.

(a) Implement the DAN model as described in section 3 in the paper. In a nutshell, the DAN model proposes to average the GloVe word embeddings to represent the sentence, and then pass this sentence representation through a multi-layer feed-forward network (or multi-layer perceptron). Your best model should get accuracy of at least 83.5% on the evaluation set (anything below will result in partial credit). Add feed-forward layers, and tune the learning rate and batch size as necessary. Include a plot of the evaluation accuracy as a function of the number of epochs.

(b) Word dropout is a popular method for regularizing the model and avoiding over-fitting. Read section 3.1 of the paper and add word dropout as described (see `pytorch` documentation), and include a plot of the accuracy of the model across different values of the dropout rate (See Figure 2 in the paper).

(c) Train 4 models with an increasing number of hidden layers (from 0 to 3 hidden layers), and compare the accuracy as the number of layers increases. Before you start training, think about when will we start seeing the effect of diminishing returns, are the results the way you expected them to be? Did the linear model outperform the model with 4 hidden layers? Include a plot of the accuracy as a function of the number of layers.

(d) Use *nn.Relu* and 2 other activation functions (of your choosing) from the torch documentation, include a plot of the accuracy across epochs. What have you learned from this experiment?

(e) For your best model, sample 5 examples from the evaluation set that the model classified incorrectly and for each example try to explain shortly why the model classified it incorrectly.

6 Topic modeling (practical) (15 points)

In this exercise, you will be asked to browse solutions for a specific downstream task in NLP. The goal of this exercise, is to get you comfortable with using others' research code and common tools. We hope that this practical experience will help you towards your final project. We encourage you to continue diving deeper, employing critical thinking and creative ideas!

In the task of **Topic Modeling** [4] we try to find some underlying semantic structure in a dataset using statistical methods. For example, we can cluster documents into topics with unsupervised methods. This tool can be used for data-analysis and data mining, with applications in NLP, Bio-informatics, Software Engineering, Social networks and so on.

Please upload the `topic_modeling.ipynb` file into colab, complete the code in it according to the instructions and answer the following question in your pdf (do not forget to submit your version of this notebook):

1. *Paper with code* is free and open resource platform with Machine Learning papers, code, datasets, methods and evaluation tables (by Meta AI Research). You can browse state-of-the-art solution to different tasks according to categories; get familiar with thousands of benchmarks datasets; compare models with varied evaluation metrics and so on.

   Explore *topic modeling on text challenge* using the task leaderboard and answer the following questions:

   (a) What are the metrics used to evaluate Topic modeling (name at least two, not including perplexity)? Please explain each of them (Feel free to use the web to extend your knowledge).

   (b) What benchmark datasets exist for this task? (name at least 3).

2. *HuggingFace* is a platform for machine learning and AI, providing open source python libraries, publicly available model weights, datasets and forums for the AI community. You will learn more about it in a dedicated tutorial in the future. In this exercises we will use and explore very specific and limited set of features from this platform: loading a dataset and a trained model.

   (a) Complete and run the code in "Loading dataset" section in the notebook by providing a reference for 20_newsgroups dataset. You can find the dataset *here*.

   (b) BERTopic is a topic modeling algorithm that combines transformers with statistical techniques (don't worry you will learn more about transformers in the future, stay tuned! You can read *BERTopic paper* and *further guides in its github* if you are interested in more details.)

   Complete and run the code in "Loading BERTopic model" section in the notebook by providing a model from Hugging Face hub. Search for a BERTopic [5] model trained on 20_newsgroups dataset *here*.

3. Now, we will evaluate the model performance on the benchmark dataset. We will use the gensim package for topic modeling evaluation. Complete and run the code in "Evaluation using Gensim"

[4] When getting into a new research subject, "review papers" are a good place to start. They provide an introduction and a literature overview (See *review1*, *review2* for example. These are intended for personal enrichment only and are not obligatory. You can access using the university account). Note that reviews are not always up to-date, especially in a field that advances as fast as NLP. Hence, searching the state-of-the-art using Google Scholar or other search tools is usually required.

[5] More about BERTopic library integration with Hugging Face in *here*

section. Note, you need to fill in the coherence metrics names (same as Q1.a). Verify the string format match the expected input to CoherenceModel according to the library documention.

Write in the pdf the results: the model, the evaluation metrics names and their values.

4. Can you think of some aspect of the task that these metrics fail to evaluate, but human-evaluator may consider? Write your thoughts in the pdf.

7 AI Disclosure & Reflection (5 points)

The following question is graded only for its completion, not its content, nor for whether you used AI or not. Furthermore, its content does not impact the grading and evaluation of the rest of your submission. The goal is to promote transparency and reflection on LLM and AI usage.

Answer the following parts briefly (1-2 paragraphs):

a) Did you use AI in the work on this assignment? If yes, please name the specific tools/models used (e.g., ChatGPT—model name, Copilot, Claude, etc.).

b) How and why did you use it – which parts of the work and for what purpose (e.g., brainstorming, outlining, code generation, debugging, editing, evaluation).

c) Reflection: What helped or hindered? What is your takeaway from this experience?

If you did not use AI, write: "No AI used", and optionally add a sentence explaining why.