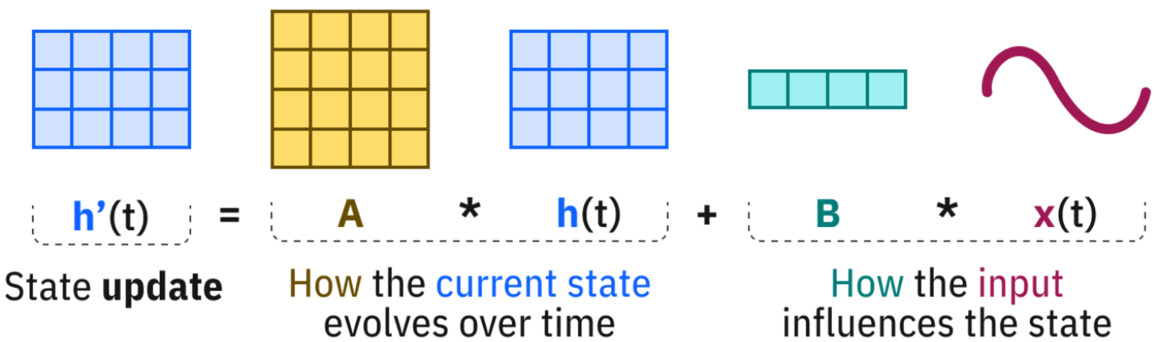
Large Language Models (Homework 2)

Due date : 2025/11/27 23:55:00 (Hard Deadline)

1 Text Classification with State-Space LLM (50%)

You will receive a dataset for a text classification task and use it to participate in a Kaggle competition. This dataset contains 14 different categories. In general, state-space model (SSM) paves a way to sequence modeling, which is seen as an alternative to implement attention mechanism. Unlike self-attention, which requires pairwise comparisons between all elements and has a computational complexity $O(L^2)$ under a sequence length L, SSM characterizes sequential dependencies through linear recurrence with a computational complexity $O(L)$. The basic computation in SSM is expressed as $h_{t+1} = Ax_t + Bu_t$ and $y_t = Cx_t + Du_t$, where the hidden state h_t evolves over time step t to generate the output sequence y_t. However, in practice, traditional SSM is not necessarily faster than attention-based model. Moreover, because their parameters remain fixed throughout the sequence, these two methods lack input adaptivity and can suffer from numerical instability during training.



Dataset description:

- This dataset includes three files: train.json, val.json, and test.json.
- The file train.csv contains 5000 entries, val.json around 2000 entries, and test.json around 2000 entries.
- Each file contains three columns: "id", "text", and "label".
- You will use the data samples in train.json to train SSM from scratch, validate the training process with val.json, and finally use test.json to evaluate the model's performance on Kaggle for the competition.

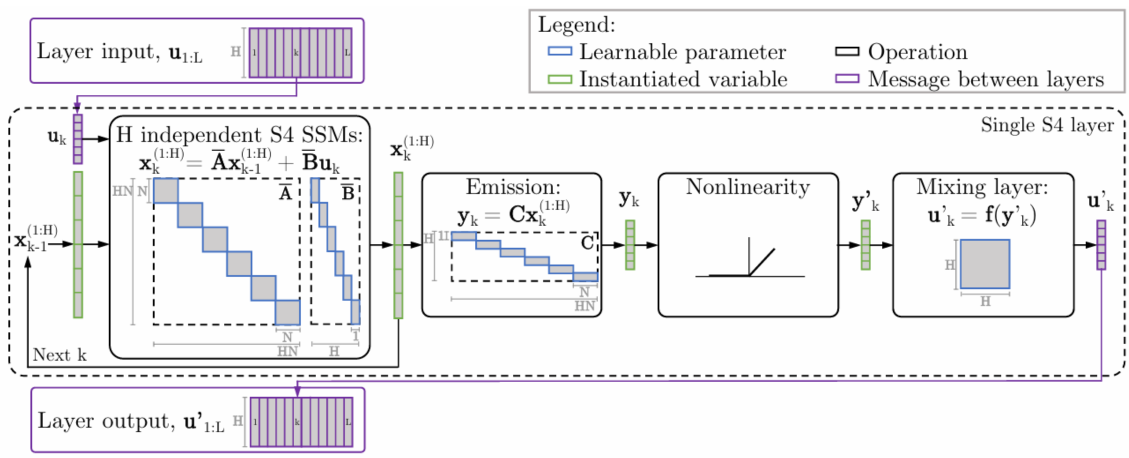
1. Please use the provided hw2_1.py file and complete the sections marked with TODO in the code. (15%)
   - The code is divided into two parts: the first part is the basic SSM, and the second part is the optimized SSM.
   - Please use the GPU provided by Kaggle to complete the code.
   - You may adjust the training parameters of the models as needed.
   - The goal is to compare the training time and performance of the two models under fair conditions.

   (a) Please refer to the provided variables and code structure to complete the implementation of the three functions in the `DiagonalSSM` class: (5%)
      - `__init__`: Initialize the model parameters, such as state dimension, continuous-time parameters, and scaling factors.
      - `_discrete`: Discretize the continuous-time state equations and compute the discrete parameters A_b, B_b, C, D.
      - `_forward`: Perform state updates and output computations based on the input sequence u using the discretized parameters.

   (b) Plot the learning curves (training and validation losses) of training data during training. (5%)

   (c) Record the training time of the basic SSM. (5%)

2. The structured state space for sequence modeling (S4) model improves upon the traditional SSM by introducing a structured and numerically stable design. It replaces the dense state matrix A with a structured (HiPPO-based) formulation to efficiently capture long-term dependencies by applying a more stable discretization method to prevent the issues when calculating the gradients, and using the precomputed convolution kernels for faster training. As a result, S4 achieves better efficiency, stability, and performance on long-sequence generation tasks compared to traditional SSMs.
   Goal: Understand how S4 addresses the issue in traditional SSMs (e.g. state-space to transfer-function conversion, fast Fourier transform, etc), then implement it by resolving the issue and compare the difference as an ablation study. (15%)



(a) Internal structure of a single S4 layer (Gu et al., 2021a) when viewed as a block-diagonal system.

   NOTE: The figure's x represents the state, and u represents the input.

   (a) You are required to understand the optimizations that S4 introduces to the SSM and implement them. Complete the TODO sections in the `FFTSSNBlock` class based on the provided parameters. (5%)

(b) Plot the learning curves (training and validation losses) of training data during training. (5%)

(c) Observe the difference in training time between S4 and the basic SSM, and discuss the possible reasons behind it. (5%)
NOTE: For more details, you may refer to the original S4 paper: *Efficiently Modeling Long Sequences with Structured State Spaces (Gu et al., 2022)*.

3. You will use the dataset to participate in the Kaggle competition. (20%)
   - Adjust the model's parameters or architecture to surpass the baseline. (5%)
   - **Leaderboard (LB) weights**: Public LB 30%, Private LB 70%. **Team name = your student ID**. (15%)

2 Low-Rank Fine-Tuned LLM (50%)

Problem Overview & Rationale

Goal: You will build a compact, task-specialized LLM for **binary commonsense reasoning** (predict exactly one token: `true` or `false`) and participate in a Kaggle competition. You will (i) fine-tune on an instruction-style training set, (ii) validate on your held-out split, and (iii) run inference on a hidden test set to produce `submission.csv`.

Why Llama 3.2 1B? It is a modern open-source LLM with strong instruction-following at a small footprint. The 1B scale fits the typical course hardware budget (single GPU), supports the fast iteration and ablations, and reduces the training/inference costs while still showing a meaningful gain from LLM adaptation.

Why LoRA? Low-Rank Adaptation (LoRA) injects the trainable low-rank adapters into the selected linear layers, keeping the base model frozen. This *parameter-efficient fine-tuning* (PEFT) greatly lowers the memory and time requirements while preserving the base model's general ability. It is modular (ship only adapters), reproducible, and ideal for controlled comparisons across various target modules.

Model Access & Hugging Face Login (required)

To use the **Meta Llama 3.2 1B** model via `transformers`, you must (1) accept the model license and (2) authenticate with Hugging Face.

1. **Request/accept access.** Visit `meta-llama/Llama-3.2-1B` on Hugging Face and click *"Access / Agree to license"*.
2. **Create a User Access Token.** Settings → Access Tokens (`hf_xxxx`, scope: Read).
3. **Log in (local dev).**

```
pip install -U "transformers>=4.44" "huggingface_hub>=0.23" accelerate safetensors
huggingface-cli login
# paste your hf_xxx token when prompted
```

4. **Log in (Kaggle/Colab).** Use a non-interactive login (env var or programmatic login) before `from_pretrained`.

Dataset

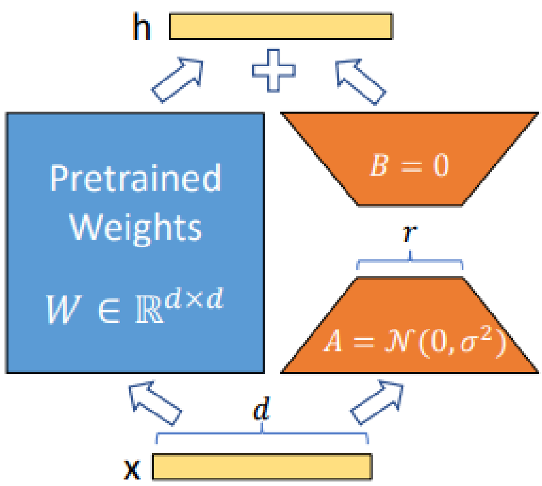
You will receive an instruction-style dataset for binary reasoning (True/False) and participate in a Kaggle competition using a LoRA fine-tuned LLM.

- We provide a single JSON file `commonsense_15k.json`; each record has `instruction`, optional `input`, and supervision fields (`output`/`answer`).

- You must split it into train/validation (80%/20% or 90%/10%). Clearly report the split ratio and the random seed.
- The hidden test set lives on Kaggle. Generate predictions with your fine-tuned model and submit a CSV file.

Background: What is LoRA?

LoRA updates a linear layer $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ via a low-rank residual $\Delta W = BA$ with $A \in \mathbb{R}^{r \times d_{\text{in}}}$, $B \in \mathbb{R}^{d_{\text{out}} \times r}$ and $r \ll \min(d_{\text{in}}, d_{\text{out}})$, so the forward becomes $Wx + \Delta Wx$. This yields far fewer trainable parameters ($\approx r(d_{\text{in}} + d_{\text{out}})$), lowers the memory/runtime, and preserves the base model while specializing for the downstream task.
NOTE: For more details, you may refer to the original LoRA paper: *LoRA: Low-Rank Adaptation of Large Language Models*



Graded Components (50%)

1. **LoRA Fundamentals concept** (2%)
   - Use the provided `finetune_lora.py` skeleton to complete a standard supervised fine-tuning pipeline on **Llama 3.2 1B**. Your code should: **load data**, **tokenize with an instruction template**, **inject LoRA**, and **train/evaluate** with HuggingFace Trainer.
   - Report:
     (a) **What is LoRA? Why low-rank?** Include the $\Delta W = BA$ formulation; discuss parameter-efficiency and deployment benefits. (1%)
     (b) **Where to attach LoRA in a Transformer?** Explain `q_proj`, `k_proj`, `v_proj`, `o_proj` (attention) and `up_proj`, `down_proj`, `gate_proj` (FFN). (1%)

2. **Target-Modules Comparison (Three Settings)** (12%) Compare the following **three** LoRA target configurations on Llama 3.2 1B:

   (a) **ATTN—light:** `["q_proj","v_proj"]`
   (b) **ATTN+FFN—medium:** `["q_proj","k_proj","v_proj","up_proj","down_proj"]`
   (c) **Full coverage—heavy:** `["q_proj","k_proj","v_proj","o_proj","up_proj","down_proj","gate_proj"]`

   - **Trainable Parameters:** Report programmatic counts (and optional estimates). (3% per 1%)

This leaderboard is calculated with approximately 31% of the test data. The final results will be based on the other 69%, so the final standings may be different.

#	Team	Members	Score	Entries	Last	Join
	challenge		0.54495			
	Strong baseline		0.51771			
	baseline		0.46866			

- **Learning Curves:** Plot **Training Loss** and **Validation Loss** per setting (separate or combined with legend). (6% per 2%)
- **Resources:** Record peak VRAM and epoch time; discuss scaling from light → heavy. (3% per 1%)

3. **Ablation and Analysis** (6%)
   - When did you obtain the **highest validation accuracy/F1**?(2%)
   - Which setting achieves the best **performance–parameter trade-off**?(2%)
   - Tie observations to module roles (`q/k/v/o` for routing vs. `up/down/gate` for semantic capacity).(2%)
4. **Kaggle Competition Submission** (30%)
   - Participate the Kaggle competition.
   - Use your best model to run inference on the **hidden test set** and upload `submission.csv`.
   - **Format:** two columns `id,answer`; `answer` must be exactly `true/false` (lowercase). Ensure IDs match the official `test.csv`.
   - **Leaderboard weights:** Public LB 30%, Private LB 70%. **Team name = your student ID.**
   - **Kaggle Scoring Policy**

Leaderboard Score Range	**Final Rank**†	**LoRA Section Grade**
score < *baseline*	–	0%
baseline ≤ score < *strong*	–	60%
score ≥ *strong and* rank 1–5	1 / 2 / 3 / 4 / 5	100% / 98% / 96% / 94% / 92%
score ≥ *strong and* rank ≥ 6	6+	90%
score > *challenge*	–	110%

† Ranks are determined by the **Private Leaderboard** (final standings). Numeric thresholds for *baseline* and *strong* baseline will be announced on the competition page.

Implementation Notes Fix random seeds; list core hyperparameters (`rank r`, `alpha`, `dropout`, `lr`, `batch_size`, `cutoff_len`, epochs). Turn off `use_cache` during training; set `padding_side="left"` and `pad_token` if missing. If using gradient checkpointing, ensure inputs require gradients. At inference, normalize outputs to {`true`,`false`} before writing `submission.csv`.
(Optional) Budget Guideline. To keep runs manageable, we *recommend* `LoRA rank` $r \leq 32$ and total trainable parameters $\leq 2 \times 10^7$. Exceeding this limit will not yield extra credit and may be flagged if it abuses shared resources.

Bonus (up to +10%)

If you can **surpass the challenge score without using LoRA**, you can earn up to **+10%** extra credit:

- Implement an alternative adaptation method (e.g., prompt-tuning, adapters, orthogonal low-rank methods, or recent PEFT/bitfit-style variants).
- Briefly cite the method's source (paper or official docs) and summarize the core idea in 5–8 sentences.
- Report your Kaggle score and show it **exceeds the challenge baseline**.

Notes and Tips

- Fix random seeds and report core hyperparameters (`rank r`, `alpha`, `dropout`, `lr`, `batch_size`, `cutoff_len`, epochs).
- Turn off `use_cache` during training and set `padding_side="left"`; define `pad_token` if missing.
- If you use gradient checkpointing, ensure inputs require gradients.
- When generating the submission, normalize outputs to the exact tokens `true`/`false`.

3 Rule

- In your submission, you need to submit two files. And only the following file format is accepted:
  - **hw2_<ProblemNumber>_<StudentID>.ipynb** file which need to contain all the results, codes and reports for each exercise (e.g. **hw2_2_0123456.ipynb**).
- Implementation will be graded by
  - Completeness
  - Algorithm correctness
  - Description of model design
  - Discussion and analysis
- Only Python implementation is acceptable.
- You may need to use the GPU for each question.
- DO NOT PLAGIARIZE. (We will check program similarity score.)