

CS 489: Introduction to Natural Language Processing

Assignment 2: Training Neural Language Models

Instructor: Freda Shi (fhs@uwaterloo.ca)
TA: Yixuan (Tom) Wang
Released: Fri, Oct 9, 2026
Due: Fri, Oct 23, 2026, 11:59 PM ET (Waterloo Time)
Weight: 9% of final grade

You may use AI tools, but you are responsible for all submitted work and must be able to explain the code and results. Using existing LSTM and Transformer implementations is allowed; implementing neural-network components from scratch is not required. No specific deep-learning framework is required.

Warning: if we find evidence that you used AI tools for anything beyond grammatical checking without declaring that use in `writeup.pdf`, you will receive 0 on this assignment.

1. Overview

In this assignment, you will build and train an LSTM and a small decoder-only Transformer, then compare their perplexity on the development set. Please only use the provided training data and initialize model weights randomly, and do not use any pretrained models or external datasets. Each model must fit on a single NVIDIA T4 GPU, as available on Google Colab's free tier, with a target training time of at most 30 minutes. See Section 8 for timing rules, allowances, and penalties.

- **Part 1 (40 points):** LSTM language model.
- **Part 2 (40 points):** Transformer language model.
- **Coding requirements (20 points):** configuration and runnable code.

Total points: 100, before extra credit.

Submit code, checkpoints, configurations, training logs, and a short report on LEARN. The TA will retrain both models and evaluate them on the test set. The three lowest TA-retrained test PPL scores in each part earn extra credit, up to 10 assignment points overall.

2. Data and Evaluation

2.1 Dataset

Use the provided English Wikipedia corpus on Hugging Face: [compling/cs489-f26-a2](https://huggingface.co/datasets/compling/cs489-f26-a2). Load it directly with the Hugging Face datasets library using `load_dataset`; the library downloads and caches the dataset automatically. Train on the train split; evaluate, tune hyperparameters, and select checkpoints on the dev split. Each dataset row is one complete document; use only its text field and preserve document boundaries. Read the dataset ID, revision, split names, and text-field name from your YAML configuration. The TA uses a private test split for final evaluation; it is not included in the public dataset.

Split	Documents	Characters	UTF-8 bytes	Access
Train	854	10,007,545	10,056,158	Released
Dev	76	1,014,702	1,021,673	Released
Test	84	1,000,953	1,005,641	TA only

2.2 Inputs and targets

Both models will use UTF-8 bytes as tokens: IDs 0 through 255 are bytes, 256 is the beginning-of-sequence token (BOS), and 257 is padding token to make the sequence length in a minibatch consistent (PAD; if needed). The vocabulary size is, therefore, 258. **The block length is fixed at 256 bytes for both models during training and evaluation and must not be changed.** Encode each document separately and split it into consecutive, non-overlapping blocks of 256 bytes, with a shorter final block if needed.¹ For a block $b_0, \dots, b_{m-1}$, use

$$\text{input} = [\text{BOS}, b_0, \dots, b_{m-2}], \quad \text{target} = [b_0, b_1, \dots, b_{m-1}].$$

A one-byte block has only BOS as its input and the single content byte as its target. **Pad short blocks on the right and ignore padding targets in the loss.** Treat each block independently: start LSTM states at zero and Transformer positions at zero. Keep all blocks, including each document's final short block.

2.3 Loss and perplexity

Train with next-token cross-entropy loss. Both models output 258 logits at each position and may use only preceding tokens to predict the target. If S is the total negative log likelihood over B non-padding target bytes, compute

$$\text{loss} = \frac{S}{B}, \quad \text{PPL} = \exp\left(\frac{S}{B}\right).$$

Compute dev PPL from the total negative log likelihood and total target-byte count over the entire dev set. Use the same inputs, targets, and metric for both models.

2.4 What you are given

- **Corpus.** The English Wikipedia train/dev corpus is available at [compling/cs489-f26-a2](#) on Hugging Face. Load its train and dev splits with `datasets.load_dataset`.
- **Example LSTM configuration.** Download [a2/lstm.yaml](#) from the course GitHub repository. Its data section specifies the Hugging Face dataset ID, pinned revision, split names, and text field. Save it as `configs/lstm.yaml`, with local output and log paths interpreted relative to that file's directory. You may use it as a reference for which parameters belong in your own configuration; the example is more detailed than necessary and is not necessarily optimal.
- **Evaluation function.** Download [a2/evaluator.py](#) from the course GitHub repository. It computes the loss and PPL using the fixed byte-level protocol.

3. Part 1: LSTM Language Model (40 pts)

Build a token embedding, a unidirectional LSTM, and a linear output layer over the vocabulary. A reasonable starting point is embedding size 256, hidden size 512, 2 layers, batch size 64, and Adam with learning rate 10^{-3} . Use dev PPL as the evaluation metric when you try alternatives, and you should only submit one final configuration that you think is best.

¹In industrial practice, training sequences are often chopped into fixed-length blocks, so less attention is paid to end-of-sequence (EOS) tokens, although they are theoretically important for modeling sequence termination.

1. (10 pts) **Model.** Implement the LSTM language model and next-token loss using Section 2.
2. (10 pts) **Training.** Implement minibatch training: shuffle training blocks each epoch, compute loss, backpropagate, and update parameters.
3. (15 pts) **Evaluation and checkpointing.** Evaluate on the entire dev set after each training epoch without computing gradients, and disable dropout and other training-only behavior during evaluation. Save the checkpoint with the lowest dev PPL throughout one training run.
4. (5 pts) **Logging and results.** In Weights & Biases (W&B), log train loss for every minibatch and full-dev loss and PPL after each training epoch. Also save these metrics locally as a CSV; its column names and layout are up to you. Plot training loss against training step and validation loss against epoch, and report the best dev PPL.

Deliverables:

- `run_part1.py` and supporting training, model, and loading code.
- The checkpoint/ directory and `config.yaml` under `artifacts/part1/` (Section 7).
- `results/part1.csv`, plus the LSTM results and W&B screenshots in `writeup.pdf`.

The TA grades the model, training, evaluation, and reported results, and evaluates a TA-retrained checkpoint for extra credit.

Tip: as a reference, the validation loss of the LSTM model with the given configuration is about 1.804 (dev PPL approximately 6.07) after training for 1 epoch. You are expected to achieve similar or lower dev PPL.

4. Part 2: Transformer Language Model (40 pts)

Build a small decoder-only Transformer with causal attention so that a position cannot attend to future input tokens. You may use an existing implementation, initialized with random weights. Use the same data, tokenizer, and loss as Part 1. Compute loss from the output logits and the targets in Section 2, without shifting the targets a second time.

A suggested starting point is 4 layers, hidden size 256, 4 attention heads, and feed-forward size 1,024. Use the fixed 256-byte blocks and vocabulary size 258 defined in Section 2. Use batch size 32 and Adam with learning rate 3×10^{-4} as initial training settings. Use dev PPL as the evaluation metric when you try alternatives, and you should only submit one final configuration that you think is best.

Complete the same four subtasks as Part 1, with the same **10/10/15/5-point breakdown**: model and loss, training, evaluation and checkpointing, and logging and results.

Deliverables:

- `run_part2.py` and supporting training, model, and loading code.
- The checkpoint/ directory and `config.yaml` under `artifacts/part2/` (Section 7).
- `results/part2.csv`, plus the Transformer results and W&B screenshots in `writeup.pdf`.

The TA grades the model, training, evaluation, and reported results, and evaluates a TA-retrained checkpoint for extra credit.

5. Practical tips for both models (optional)

These suggestions may help with implementation and tuning; they are not additional grading requirements.

- **Place dropout appropriately.** For LSTMs, try dropout after embeddings, between stacked layers, and before the output layer. Check where your library applies dropout; an inter-layer dropout setting does not add dropout after the final layer. For Transformers, consider embedding, attention, and residual dropout. Try rates of 0.1–0.3 and select using dev PPL; more dropout is not always better.
- **Clip gradients for stability.** To limit exploding gradients, especially with LSTMs, try clipping the global gradient norm to 1.0 after backpropagation and before updating parameters.
- **Refer to online examples, but adapt them.** Consult your chosen library’s documentation and examples. Cite external code, initialize weights randomly, and adapt examples to byte tokens and independent blocks, resetting LSTM states for every block. Some causal language-model implementations shift targets internally; compute loss from logits and this assignment’s targets to avoid shifting twice.
- **Overfit a tiny training subset first.** Check that loss drops substantially on a few fixed minibatches, temporarily disabling dropout if helpful. Test training, saving, reloading, and evaluation before a full run.
- **Use train/dev curves to guide tuning.** If train loss falls but dev loss rises, try stronger regularization or a smaller model. If both stay high, check the implementation, learning rate, and optimization before increasing dropout.
- **Tune within the training budget.** Measure one epoch’s runtime and GPU memory use on a T4 before choosing model size and epoch count. Choose settings targeting the 30-minute per-model budget, subject to the timing allowances and penalties in Section 8.

6. Coding Requirements (20 pts)

- (10 pts) **Configuration (the no-hard-coding rule).** Put model dimensions, learning rate, batch size, epoch count, random seed, file paths, and other tunable settings in YAML so they can be changed without editing Python code. If you are unsure whether a setting belongs in YAML and might change it, put it in YAML. The token IDs, 256-byte block length, and evaluation protocol in Section 2 are fixed, not tunable settings. Save the settings actually used as `config.yaml` in each model’s artifact directory. Interpret local paths relative to the YAML file’s directory, and rewrite relative paths when exporting the configuration to its new location. The exported configuration must work after the submission directory is moved to another machine; do not save paths specific to your own machine. Use this same file to reproduce the submitted training run.
- (10 pts) **Runnable code.** Pin the Python version and dependencies with `uv`, provide the training and loading entry points, and document setup and reproduction commands in `README.md`.

The following commands must work from a fresh environment. For online W&B logging, replace the API-key placeholder below only in your local shell.

```
uv sync
export WANDB_API_KEY="<your-wandb-api-key>"
uv run python run_part1.py --config artifacts/part1/config.yaml
uv run python run_part2.py --config artifacts/part2/config.yaml
```

The TA will use their own W&B key and does not need your API key.

Warning: do not include a W&B API key anywhere in your submission, including source code, configuration files, logs, screenshots, or the README. Submitting a W&B API key will result in a deduction of all 20 points for Coding Requirements.

You may use [TruffleHog v3.99.2](#), which includes W&B API-key detectors, to help check your submission. After installing it, run the following command from the extracted submission directory; these options disable online key verification:

```
trufflehog filesystem . --no-verification \
  --results=unverified --fail
```

Detection is not guaranteed, so also manually check all submission files, including logs, screenshots, and your report.

7. Submission

Submit a single ZIP on LEARN with the following required files and directories. Include any supporting Python modules; their names and layout are up to you.

```
a2/
  pyproject.toml
  uv.lock
  README.md
  run_part1.py
  run_part2.py
  submission.py
  evaluator.py
  <supporting Python modules>
  artifacts/
    part1/
      config.yaml
      checkpoint/
    part2/
      config.yaml
      checkpoint/
  results/
    part1.csv
    part2.csv
  writeup.pdf
```

7.1 Checkpoints

Save only each model's best-dev checkpoint in its `checkpoint/` directory, using your framework's checkpoint format and file names. The accompanying `config.yaml` must identify the model type and settings needed to rebuild it. Document an exact loading and evaluation command in `README.md`.

7.2 Logging and short report

Use a separate W&B run for each training run and record its configuration. Log the train loss for every minibatch, using cumulative minibatch count as the training step. After each training epoch, evaluate the entire dev set and log its loss and PPL together with the epoch number, starting at epoch 1. Compute these dev metrics from the total negative log likelihood and non-padding target count across all dev minibatches. Use training step as the horizontal axis for training loss and epoch as the horizontal axis for validation loss. In W&B, associate the validation metrics with your logged epoch field as their step metric; do not reset the global logging step to the epoch number. Support

W&B offline mode so the TA can run your code without an account or API key. Also save these metrics locally in `results/part1.csv` and `results/part2.csv`. You may choose the CSV column names and layout.

Submit a report of at most two pages (`writeup.pdf`), including W&B screenshots of training loss versus step and validation loss versus epoch for each model, a table of their best dev PPL, and a brief comparison of the results. W&B run links are not required. State the model sizes and training settings used. In the report, declare any AI usage beyond grammatical checking, naming the tools and describing what they helped you do. AI usage limited to grammatical checking does not need to be declared. Include only the files and directory contents specified in the submission layout above in the ZIP; do not include any other files.

8. TA Evaluation and Extra Credit

The TA retrains both models from random initialization using your YAML configurations and evaluates them with the supplied `evaluator.py`. For each model, the checkpoint with the lowest dev PPL throughout its TA training run is evaluated on the private test set. Each submitted training run must fit on a **single NVIDIA T4 GPU** (as available on Google Colab's free tier), with a target runtime of **30 minutes per model**. Wall-clock timing starts when the training program's Python process starts and ends when it exits, including imports, data loading or downloading, preprocessing, training, dev evaluation, checkpoint saving, and logging. Environment installation with `uv sync` happens before the training program starts and is not timed. A TA run that finishes within **40 minutes** incurs no timing penalty. A TA run that takes more than 40 minutes incurs a deduction for the corresponding part. If it has not finished within **60 minutes**, the TA stops it and awards **0 points for that part**, with no extra credit for that part. Choose the epoch count and other settings in your YAML configurations to target the 30-minute budget rather than relying on the allowance. Small numerical differences between repeated runs are normal.

In each part, the three lowest TA-retrained test PPL scores earn **10, 8, and 5 assignment points** for first, second, and third place, respectively. Extra credit is capped at **10 assignment points overall** across both parts. The deduction amount for runs exceeding 40 minutes, tie rules, and reproducibility tolerances will be announced before release.

9. Policies

Collaboration and academic integrity

All tasks are **individual work**. You may discuss high-level ideas, but do not share or copy another student's code, results, or report. Cite external code, declare AI assistance beyond grammatical checking in `writeup.pdf`, and submit only genuine training results and model artifacts.

Late policy

The deadline is **Fri, Oct 23, 2026 at 11:59 PM Eastern Time**. The universal **72-hour grace period** accepts submissions without penalty until Mon, Oct 26, 2026 at 11:59 PM Eastern Time. No submissions are accepted after the grace period ends. Extensions beyond the grace period are considered only for medical emergencies: a valid Verification of Illness Form (VIF) must be submitted before the original deadline and cover both that deadline and the entire grace period.

Useful documentation

See the [uv documentation](#) for environment setup and the [W&B documentation](#) for logging. Consult your chosen framework's documentation for model configuration, training, and checkpoint loading.