Assignment A1 : Tokenization and Statistical Language Modelling

Individual assignment • 20% individual-report component Released: 19 Nov 2026 Due: 26 Nov 2026, 23:59

You will build a simple statistical (n-gram) **language model**, use a provided **tokenizer** to cut text into tokens, and compare your model against a large neural language model (Gemma 4, 31B parameters) served on **TULIP**. The question to answer: *how do you compare two language models that cut text into different tokens, and what does perplexity actually tell you about a model?*

From the lecture

Hands-on companion to *Lexical Analysis + Language Modelling* (19 Nov).

At a glance — input, output, how it's evaluated

Input	the training text and your 300 held-out paragraphs
Output	`a1_results.json`, `samples.txt`, `findings.md`, `reflection.md`, your code
Evaluation	5 points auto-checked against a per-student reference; 5 points by a TA on your written answers.

1. What you write (about 25 lines)

Three functions. Check them first with `python selftest.py` — a toy corpus with hand-worked answers, no network needed. The precise contract is in the **Reference** (page 2).

1. **The n-gram model** (`fit`, `prob`). A bigram over the provided tokens with **add-k** smoothing. The kit picks k for you on a dev part of the training text.
2. **Scoring with TULIP** (`doc_nll`). The kit returns the log-probability of every token of a paragraph under TULIP; you turn those into the paragraph's negative log-likelihood.
3. **Sampling** (`sample`). Generate one continuation from your n-gram (at most 40 tokens, temperature 1.0).

Optional, never graded: a trigram (`-order 3`) and/or Kneser–Ney smoothing. Your n-gram counting must be your own code (no `nltk.lm`, kenlm or similar).

2. What the kit does for you

- **Tokenizer** — trains a **byte-level BPE** tokenizer (vocabulary 8000) on the training text. Byte-level is *lossless*: no `[UNK]`, no dropped whitespace.
- **Perplexity**, two ways — **per token** and **per byte**, for both models, on your 300 held-out paragraphs. The tokenizers differ, so per-token numbers are on different scales; *per byte is the fair comparison.*
- **Vocab sweep** — repeats the n-gram part at vocabulary **257** (pure bytes), **1000** and **8000**. This is the evidence for your findings.
- **TULIP** — scores your paragraphs and generates continuations; writes `a1_results.json` and `samples.txt`.
- **Submission** — `make_zip.py` builds the zip and runs the format checker.

Run everything with `python a1.py -student-number <yours>`: about 2–3 minutes on a laptop,

no GPU. Re-runs reuse the cached TULIP results, so after your first successful run you can work offline.

3. Findings (`findings.md`, 2–4 sentences each)

Answer under the fixed headings in `findings.md`, using the numbers your run printed. Arithmetic lines do not count towards the sentence limit.

Q1. Rank your 257-vocab n-gram, your 8k n-gram and TULIP by **per-token** perplexity, then by **per-byte** perplexity, quoting the numbers. Why do the two rankings disagree, and which one is the fair comparison? Why is the gap between TULIP's per-token and per-byte perplexity so much larger than for your n-gram?

Q2. Compare your 8k n-gram and TULIP **per byte**, quoting both numbers. Is the result what you expected from a bigram model versus a large neural model? Give two plausible reasons, and support each with evidence from your own run — for example the tokens one model finds most surprising (`a1.py` prints the 10 most surprising tokens of each model), or how the generated text reads.

4. Reflection (`reflection.md`, max 200 words)

Answer these three, plus the AI-use declaration. Graded for honesty and insight, not length.

1. **What did you know before?** — your starting point on this topic.
2. **What was hardest, and how did you get unstuck?**
3. **Read your `samples.txt`.** Whose continuations read better, your n-gram's or TULIP's? Does that agree with their per-byte perplexities? What does this tell you about what perplexity measures, and what would you try next?

AI-use declaration. AI tools (including TULIP's `code` model) may help you write code; state which parts (if any) were AI-generated and *how you verified them*. Your numbers are recomputed, so you must be able to explain every number you submit.

5. Grading (10 points)

Correctness — automatic (5). Your numbers are recomputed on *your own* held-out set, so there is one correct answer per student (nothing is compared to a classmate or a shared key):

- n-gram perplexity, per token and per byte, recomputed from your `tokenizer.json` and settings — **2**
- TULIP document scoring, within tolerance (±1% per byte; partial credit beyond) — **2**
- the vocab sweep and the byte/token counts — **1**

Understanding — TA (5). Graded on your written answers, which must use and correctly interpret *your own* numbers (generic explanations score low):

- Findings Q1–Q2 — **3**
- Reflection (including the AI-use declaration) — **2**

6. How to submit

What. One file `A1_<studentnumber>.zip` containing: `a1_results.json`, `tokenizer.json`, `sweep/`, `samples.txt`; `findings.md` (your answers to Q1–Q2); `reflection.md` (reflection + AI-use declaration); and `code/` (the code that produced your numbers). Do *not* include data, your `.env` file, or the TULIP cache.

How. Run `python make_zip.py <studentnumber>` — it builds the zip and runs the format checker. Fix every error (`x`) before you upload; warnings (`!`) are only hints. Upload the zip to **Brightspace**. Use the *same* student number you passed to `load_lm` — your held-out set, and therefore your grade, depend on it.

Reference

Data. We use **WikiText-103**, a collection of Wikipedia articles[1]. The starter kit ships a fixed sample of it, so no data is downloaded: `load_lm(student_number)` returns a **training set** of 5000 paragraphs (the same for everyone) and **your own held-out set** of 300 paragraphs, drawn from a fixed pool of 500 using your student number. A paragraph is one non-heading line of WikiText-103 (150–1200 characters), kept exactly as in the raw data.

Definitions (the grader uses exactly these)

- **Paragraphs** are scored *independently*: no concatenation, no separators.
- ***n*-gram.** Tokens are `tok.encode(p).ids`. Pad the start with one `<s>` (two for `-order 3`); `<s>` is context only, never predicted. Every paragraph token is predicted; there is no end token. V is the tokenizer's vocabulary size. **Add-k:** $P(w \mid h) = \dfrac{c(h,w)+k}{c(h)+kV}$; a history never seen in training gets $P = 1/V$.
- **TULIP.** The kit sends each paragraph with TULIP's start-of-text token as context. Every paragraph token is scored, including the first; no end token.
- **Perplexity.** Natural-log NLL summed over all scored tokens of your 300 paragraphs. $\mathrm{PPL}_{\text{token}} = \exp(\mathrm{NLL}/N_{\text{tokens}})$, with N_{tokens} counted by *that model's* tokenizer; $\mathrm{PPL}_{\text{byte}} = \exp(\mathrm{NLL}/N_{\text{bytes}})$, with N_{bytes} the UTF-8 bytes of the paragraphs exactly as loaded (the same for both models). Hence $\ln \mathrm{PPL}_{\text{byte}} = \ln \mathrm{PPL}_{\text{token}} \cdot N_{\text{tokens}}/N_{\text{bytes}}$.
- **TULIP's model.** `chat` is **Gemma 4 31B**, an *instruction-tuned* (chat) model served with 4-bit weights.[a]

[a] Checkpoint `google/gemma-4-31B-it-qat-w4a16-ct`, served on TULIP under the alias `chat`.

Using TULIP (TU Delft's LLM service)

- You need to be on the TU Delft network or **eduVPN**, with your personal key in a `.env` file (the kit's README shows how). Never share or commit your key.
- Start with `python tulip_check.py`: it checks VPN, key, model, generation and scoring, and tells you how to fix anything that fails. Do this **early** — not on the evening of the deadline.
- All TULIP calls go through the kit's client: one paragraph per request at a polite pace, with caching and polite retries, stopping if a run would exceed 400,000 tokens (well within your personal TULIP quota (see Brightspace)). Please do not write your own request loops.
- TULIP's documentation is at https://docs.tulip.tudelft.nl/ (also only reachable on the TU Delft network or eduVPN). If TULIP is unavailable near the deadline, your cached results stay valid; report the outage on the course forum.

[1] HuggingFace dataset `Salesforce/wikitext`, configuration `wikitext-103-raw-v1`; licence CC BY-SA 3.0.