COMP 4901B Large Language Models Fall 2025 Course Logistics and FAQ

1. Prerequisites and Materials

Students are required to be familiar with programming, machine learning basics and deep learning basics. Specifically, for HKUST students, you are required to have taken one of the machine learning courses COMP3211, COMP4211, or COMP5212, and enroll COMP4211 in this semester if you only took COMP3211 before.

2. Honor Code

Students are required to work on the homework **independently** unless otherwise specified.

Use of Generative AI tools: we allow use of generative AI tools such as ChatGPT and Claude to help the coding part in your homework, but for any writing part of the homework, you are not allowed to directly copy outputs from generative AI tools.

We have **zero tolerance** on honor code violation – any single time of violation will cause you to fail the course directly.

3. Grading

- Attendance (10%)
- 4 assignments (40%=4*10%)
- Mini-project (25%)
- Final exam (25%)

Attendance: there will be occasionally in-course quizzes to record attendance, 80% of the attendance will give you full score on this. For example, suppose we have 10 such quizzes across the semester, you can miss 2 of them and still get the full score. The correctness to the quiz questions will not influence the score.

Assignments: there will be 4 assignments, all of which are programming-centric and may require writing a simple report. These assignments will be released on an ongoing basis.

Mini-project: The mini project will be a group project with 2-3 people for each group.

Exams: Exams are close-book.

4. **Submitting Assignments**

   Assignments release, submission, and grading will be through Canvas unless otherwise specified.

5. **Late Assignments**

   Each student will have a total of three free late (calendar) days to use for homeworks. Once these late days are exhausted, any assignments turned in late will be penalized 20% per late day. However, no assignment will be accepted more than three days after its due date. Each 24 hours or part thereof that a homework is late uses up one full late day.

6. **Lecture Video Policy**

   Lecture videos are available and will be released on Canvas.

FAQ

1. How should I ask for TAs to help me debug code? (courtesy of Stanford CS229)

    a. Please note that the teaching staff will not debug code longer than 2-3 lines via Canvas. Learning to debug is a critical skill for software programmers. The TAs are **discouraged** from helping you look at and debug large blocks of your code during the office hours.
    b. The best way to use office hours and ask TAs for coding questions would be
        i. You should come to office hours having done your own legwork and ruled out basic logical errors. Identify the place where the error is suspected to come from by doing ablation studies. (Please see below for some common debugging tips.)
        ii. During the office hours, you should articulate what your goals are and what you have observed in your experiments, what you think might be the problem, and what advice you need to move forward.
        iii. The TAs will mostly help you by **looking at and analyzing the outputs of your code** instead of looking at the original code. Typical advice that the TAs might give you would be to ask you to do more analytical or ablation studies about your code. For example, when you observe that your test error does not decrease as training for longer, the TAs might ask you to check if your training error decreases. If your training error does not decrease, then the TAs might ask to check if the gradient of your algorithm is implemented correctly.

c. Here are some common debugging strategies that might be useful (courtesy of Stanford CS221)
    i. Construct small test cases that you have worked through by hand and see if your code matches the manual solution.
    ii. Spend some time understanding exactly what the test cases are doing and what outputs they are expecting from your code.
    iii. If possible, write your codes in small chunks and test that each part is doing exactly what you expect.
    iv. PDB is the default python debugger. It is very helpful and allows you to set breakpoints. You can set a breakpoint with the following lines: import pdb; pdb.set_trace() .
    v. Printing the state of your computation frequently can help you make sure that things are working as expected and can help you narrow down which portion of your code is causing the bug you are seeing, e.g. print("var1 has current value: {}".format(var)) .
d. Debugging tips for timeouts:
    i. Set operations in general are pretty slow, so if you have any see if you can do them in some other way.
    ii. Check if all loops / linear operations are necessary. For example, with searching through a list for a specific item, sometimes you can make that constant time by giving each item an ID (say 0, 1, 2, 3) and then using a dictionary as a cache (although sometimes you just have to live with the cost).
    iii. If you have a specific helper function you are calling a lot, see if there is anything in there you can optimize!
e. Other debugging tips
    i. If you do not know what type a variable is, use type(.) .
    ii. If you are running into issues where "None" pops up, a function may not be returning what you are expecting.
    iii. For indexing into lists: example_list[a: b] is INCLUSIVE for a but EXCLUSIVE for b.
    iv. If a function has optional arguments, make sure you are feeding in the proper arguments in the proper places (very easy to mess up).
    v. Since python 3.6, you can use f-strings for printing debug messages, rather than format.
    vi. Because of broadcasting and other implicit operations, it's useful to assert shapes of np arrays (and tensors for deep learning) after each operation that can change the shape.