

Course Information

Instructors: *Alex Andoni*, *Edo Liberty*

1 Basic Information

Lectures:

- Time: MW 2:40–3:55pm.
- Location: 313 Fayerweather.

Instructors:

- **Alex Andoni** (andoni@cs.columbia.edu)
- **Edo Liberty** (le2457@columbia.edu)

Teaching Assistants:

- **James Wang** (jw5002@columbia.edu)
- **Rushin Bhatt** (rsb2213@columbia.edu)

Website: There is no textbook. Lecture notes, scribes, and links to papers and other resources are posted on the class website:

<http://www.cs.columbia.edu/~andoni/algof26/>

Office hours: see calendar linked from Courseworks (requires LionMail login).

Courseworks: Class announcements, including homework assignments and project deadlines, will be posted on Courseworks.

Edstem: There is also an Edstem forum set up for the class (accessible through Courseworks). You are encouraged to discuss class lectures and related topics, as well as ask questions or clarifications. (But you *cannot* discuss homework solutions on Edstem.)

2 What This Class Is About

Large Language Models (LLMs) are built from a pipeline of stages: data collection and filtering, tokenization, retrieval, the transformer architecture itself (attention in particular), training via stochastic optimization, compression (quantization, sparsity), and inference under a memory/latency budget. Each stage hides one or more algorithmic problems. This class is about those algorithms, and about the theory behind them.

The focus of the class is on *why*, not *what*: not a catalog of the techniques currently used, but the fundamental reasons the good techniques look the way they do. The class is not a recipe; the recipes change every year, the fundamentals do not. Concretely, for each stage of the pipeline we will ask:

- *What is the problem?* Identify and define the computational problem hiding inside the stage.
- *What is the framework?* Build a formal model in which one can rigorously reason about the problem, and define what a solution costs (time, memory, communication, bits, samples, ...).
- *How to measure progress?* Analyze algorithms within the framework, and compare them.
- *What are the bottlenecks?* Identify what cannot be improved (lower bounds, hardness), and hence what it would mean to improve further.

This class is focused on rigorous analysis, including discussing formal definition of the models and of the costs, an algorithm's analysis of its runtime and a proof of its correctness (or of its approximation guarantee). Sometimes we will be able to do only one of the two, or only in a restricted setting; we will be explicit about which.

What the class is not about: while we will try to cover most stages of the LLM pipeline, the main focus is on the stages where algorithms have theoretical underpinnings. There is little emphasis on implementation/code (though it can appear in the final projects), and less emphasis on the pure Machine Learning perspective (e.g., generalization).

At the end of the class, expect to:

- know the algorithmic problems that arise at each stage of the LLM pipeline, and the formal frameworks in which they are studied;
- be able to read research papers in the area, and judge what their claims mean and whether they are justified;
- be able to formulate a new algorithmic question arising in LLMs, and analyze it rigorously.

Tentative topics. Roughly two lectures per topic:

- Introduction to language models and transformers;
- Data and tokenization;
- Nearest Neighbor Search: retrieval (RAG), and algorithms;
- Attention mechanism;
- Randomized linear algebra;
- Quantization;
- Mixture of Experts;
- Optimization.
- Expressivity: transformers as a computational model;
- Alignment;
- Privacy and provenance.

3 Prerequisites

First and foremost, mathematical maturity is a must: the class is based on theoretical ideas and is proof-heavy. You are expected to be able to read and write formal mathematical proofs, at the level of COMS 4231 (Analysis of Algorithms). More specifically:

- **Math:** probability (linearity of expectation, variance, concentration bounds), and linear algebra (eigenvalues/eigenvectors, matrix norms, SVD).

- **Algorithms:** you should appreciate what it means to design an algorithm and to analyze it: order notation for runtime, correctness proofs, NP-hardness. COMS 4231 or equivalent is recommended, but not required if you have a solid math background.
- **Machine Learning:** not much is needed—less than for a typical ML class. We will introduce what we need (e.g., what a transformer is) from scratch, from an algorithmic viewpoint.
- **Coding:** not needed. It is fine to do only theory in this class, including in the final project. Implementation-type projects are welcome too; for those, you may use coding agents.

Undergraduate students and students from other departments are welcome, provided they have the background above.

4 Evaluation and Grading

Your grade is based on the following three components:

- Final project: 50%;
- Scribing (one lecture, plus an extra chapter): 20%;
- 1 homework (followed by a meeting with a TA): 30%.

5 Homework

There will be one homework, posted on Courseworks. It will contain a few problems, testing that you can work with the formal frameworks from class. Please follow the Homework Submission Guidelines below.

Meeting with the TA. After submitting the homework, you will meet with a TA (individually, ≈ 10 – 15 minutes) to explain your solutions. The purpose is to check that you understand what you submitted: expect to be asked to walk through an argument, and to answer follow-up questions about it. Your homework grade is based on both the written solution (graded as pass/fail) and this meeting (a numerical score). Sign-up details will be posted on Courseworks.

Late policy. You have a default 5 days of extension (fractions of a day are rounded up). Once you’ve used up the 5 days, a late homework will be penalized at the rate of 10%, additively, per late day or part thereof (i.e. fractions of a day are rounded up), for up to 7 days. A homework submitted more than 7 days after the deadline will not be accepted. Exceptions will be made only for exceptional unforeseen circumstances (e.g., serious illness), in which case you will need to provide some additional documentation (e.g., doctor’s note).

You are strongly encouraged to start working on the homework *early*: some problems may require you to sit on the problem for a while before you get your “aha” moment. Starting early also gives you time to ask questions and make effective use of the office hours of the teaching staff.

Writing up solutions: precise and formal proofs. The goal of the class, in part, is for you to learn to reason about algorithms, precisely describe them, and formally prove claims about their correctness and performance. Hence, it is important that you write up your assignments *clearly, precisely, and concisely*. Legibility of your write-up will be an important factor in its grading. When writing up (algorithmic) solutions, keep in mind the following:

- The best way for you to convey an algorithm is by using plain English description. A worked example can also help; but revert to pseudocode only if necessary. Generally, give enough details to clearly present your solution, but not so many that the main ideas are obscured.
- Unless stated otherwise, the analysis of the algorithm has to include both 1) proof of correctness, and 2) upper bound on performance (usually runtime, but sometimes space as well).
- You are encouraged (but not required) to type up your solutions using LaTeX (e.g., using Overleaf).

Note that our lectures will generally be at a slightly lower level of formalism, in the interest of time.

Homework Submission Guidelines:

- Submit your homework electronically via Gradescope. You may write solutions by hand, in which case you should either scan or photograph your solutions.
- The homework is due on the specified due date 40 minutes before the class starts (2pm).

6 Scribing

You will scribe one lecture, possibly jointly with one or two of your colleagues (depending on the final enrollment). Scribe notes have two parts:

- **The lecture.** A complete and self-contained write-up of the lecture: definitions, statements, and full proofs (including the details that were skipped in class), in the rigorous style described above.
- **An extra chapter.** A section on extra material related to the lecture that was *not* covered in class. Think of this as “if the lecture had extra 1h, what would one cover of interest”. Examples: a follow-up or related result, an alternative algorithm or proof, a lower bound, a connection to how the idea is used in practice, or a recent paper. Explain it in your own words, at the same level of rigor as the rest of the notes ($\approx 2 - 3$ pages). You can choose the topic yourself (based on suggested readings from the class webpage), or ask the staff for suggestions.

You have to use the LaTeX template available on the class website. Scribe notes are due one week after the lecture. The staff will review the notes and, if necessary, ask you to revise them; the notes are then posted on the class website for everybody to use.

7 Collaboration and Academic Honesty

Collaboration: you are permitted to discuss the homework *assignment*. If you collaborate, you must write the solutions *individually* (without looking at anybody else’s solutions), and acknowledge anyone with whom you have discussed the problems. It will be considered an honor code violation to consult solutions from previous years, from the web or elsewhere, in the event that homework problems have been previously assigned or solutions are available elsewhere.

You are expected to abide by the policies of academic honesty. The CS department web page lists the department’s academic honesty policies: <http://www.cs.columbia.edu/education/honesty>.

8 Use of AI/LLMs

This is a class about LLMs, and you are allowed to use AI/LLMs as a resource in it. However, you should use them wisely. A large part of the class grade requires you to demonstrate essential understanding of

the material *yourself*: the meeting with the TA after the homework, the project presentation, and the scribe notes (which you have to be able to defend). So you shouldn't become too dependent on the LLM.

Treat the homework as a low-stakes way of getting practice with thinking and learning the material. We can't stop you from applying AI/LLMs to blindly solve the problems, but you will not be able to explain the solutions in the TA meeting, and your grade will suffer.

Here are suggestions for how to use AI/LLMs in a smart way, especially for the homework.

Try to solve the problem yourself. Identify the relevant lecture material, and try to figure out an approach. Talk with a classmate (per collaboration above).

If you end up using the AI, don't just copy-paste the question/answer. First, you can enter a prompt like the following: *I'm a student trying to learn algorithms with theoretical guarantees, specifically [topic], and I'd like you to be a helpful study partner who doesn't just give me the answer, but helps me learn and solve problems myself.*

Then, you can use prompts like: *"Explain the analysis of Locality Sensitive Hashing, like I'm a beginner."*

"I'm trying to solve the following problem: [insert general description]. However don't tell me the answer. But my general approach is..."

"Here's my derivation. Can you give me a hint as to the first line that breaks?"

You must understand and verify everything you submit. A way to test whether you understand: could you explain your answers to a peer without relying on any external aid? (For the homework, this is exactly what the TA meeting checks.)

For the final project and the scribe notes, you may use AI tools (including coding agents for implementation projects); briefly state in the write-up how you used them. You remain responsible for the correctness of everything you submit.

Ultimately, you're taking this class because you want to learn how to design and think about modern algorithms. The AI already knows the material pretty well, so don't let it rob you of the opportunity to learn.

9 Final Project

In the final project you will delve into a particular topic in more detail, in a team of your own. The final projects can be of the following types:

- Research/Exploratory (theory): investigate a research question on your own, e.g., develop an algorithm for some stage of the LLM pipeline and prove its guarantees; prove a lower bound or an impossibility result; or formalize a problem that is currently handled only heuristically, and analyze the heuristic within your framework.
- Implementation/Empirical: implement some of the algorithms from the class (or from the theoretical literature), and test them, ideally against the heuristics used in practice, on real models or datasets. One aspect of such projects should be a comparison among a few algorithms, and an explanation of what the theory predicts versus what you observe. You may use coding agents for the implementation.

Teams: you are allowed to have a team of 1–3 people in total per team. Single-person teams are

possible but discouraged and need special permission from the instructors¹.

Topic: the topic of your project must be within the scope of the class: algorithms for some part of the LLM pipeline, with the focus on provable guarantees (for the implementation type, you may compare such theoretical guarantees with heuristics). More details and suggestions will be given later in the class. You are encouraged to find a team early, and discuss potential topics with the instructors.

Timeline:

- *Project proposal* (October): about 1 page, describing the question, why it is interesting, and what you plan to do.
- *Progress report* (November): 2–3 pages, describing what you have done so far, what worked or did not, and the plan for the rest.
- *Presentation/poster*: during the last lectures.
- *Final write-up*: due during the finals period.

See the calendar below for the exact dates.

¹The reason is that the topics are hard, and having a collaborating partner/s will qualitatively improve your understanding of the topic and experience overall.

10 Calendar (tentative)

Below is the schedule of lectures and assignments. Information regarding what each lecture covers will be periodically updated on the course website.

Lecture	Date	Topic	HW out	Due
1	9/9	Intro to LMs and Transformers		
2	9/14	Data, Tokenization		
3	9/16	Data, Tokenization		
4	9/21	Nearest Neighbor Search: RAG		
5	9/23	Nearest Neighbor Search: Algorithms		
6	9/28	Attention Mechanism		
7	9/30	Attention Mechanism		
8	10/5	Randomized Linear Algebra		
9	10/7	Randomized Linear Algebra		
10	10/12	Quantization		
11	10/14	Quantization		
12	10/19	Mixture of Experts		
13	10/21	Mixture of Experts		
14	10/26	Optimization		Project proposal due
15	10/28	Optimization		
	11/2	NO CLASS: Academic holiday		
16	11/4	Expressivity: Transformers as a Comp. Model		
17	11/9	Expressivity: Transformers as a Comp. Model		
18	11/11	Alignment		
19	11/16	Alignment		
20	11/18	Privacy and Provenance		Progress report due
21	11/23	Privacy and Provenance		
	11/25	NO CLASS: Thanksgiving		
22	11/30	tbd		
23	12/2	tbd		
24	12/7	tbd		
25	12/9	Final project presentations/posters		
26	12/14	Final project presentations/posters		
	12/21	Final project write-ups due		Project due