

# Syllabus

## 705.743: ChatGPT from Scratch: Building and Training Large Language Models

**Wednesdays, 4:30 p.m. - 7:35 p.m**

### Instructor Contact

#### **Ted Staley**

Work: 240.302.7138

Cell: 314.882.5648

E-mail: [edward.staley@jhuapl.edu](mailto:edward.staley@jhuapl.edu)

I prefer that students contact me via email. Please be sure to include course number in the subject line. I will make every effort to reply within 48 hours. If an issue is urgent, please indicate so in the subject line and I will make it a priority to respond as soon as is practical.

### Office Hours via Zoom (Time TBD)

This course will use Zoom to facilitate weekly, synchronous office hours. You are not required to participate in Office Hours; however, you may find them very beneficial for receiving more timely answers to questions related to the course content and assignments.

During the first week of the course I will set up regular Office Hours and provide links. Students will click that link or the Office Hours link on the course menu to access Zoom and participate. You are encouraged to post any questions you would like to have answered during the live Office Hour sessions to the Office Hours Discussion Board by the night before. Recorded Office Hour sessions will be posted to Canvas for any students who were unable to participate in the "live" sessions or for students who like to listen to them again.

For more information regarding Zoom, please see the [Zoom Student Quick Start Guide](#).

### Course Description

Large language models (LLMs) like ChatGPT have ushered in a new wave of virtual assistants, chatbots, and text generators, and many see them as a paradigm shift in how humans interact with machines. Huge development ecosystems have arisen around LLMs, often abstracting away how they work in an effort to make them accessible to more people. While the democratization of this technology is important, LLMs cannot be fully harnessed and improved without understanding their inner workings at a fine level. In this course, students will build a small version of a text generation model like GPT3 over the course of several weeks. They will learn about the details of the GPT architecture from bottom to top, how it came about, and how it is used today in applications like ChatGPT. Once these fundamentals are established, students will build their own research experiment on top of their home-grown language models. After this course, students will be prepared to build and modify language models for their own purposes and dive into LLM research or novel applications.

### Prerequisites

**EN.705.643 Deep Learning Developments with PyTorch** or equivalent practical experience using PyTorch. Mathematical foundations of neural networks highly recommended (i.e. EN.605.742) or linear algebra.



## Course Goals

To understand the inner workings of large language models (LLMs) and gain experience implementing them first-hand, albeit at a small scale. Understand additional technologies which are used to turn a pretrained language model into a powerful piece of software like ChatGPT, and how this process could be used for specific applications of interest. Finally, explore one of these additional technologies or a specific LLM research question in a final project.

## Course Objectives

By the end of this course, you will be able to:

- Implement a small language model on your own
- Understand how you might approach training and inference at scale, if of interest to you
- Understand how to use LLMs for specific tasks and applications
- Understand how to edit existing models through various fine-tuning techniques
- Understand some of the key research questions in LLMs, and the current state-of-the-art models
- Use LLMs in your own projects, applications, and research

## Course Structure

The course is 12 weeks long (summer) and divided into two parts:

The first seven weeks of the course will focus on the details of the language model architecture and implementing language models in PyTorch. During this period, weekly modules will be used to focus on one component of language models at a time. Each module will have (1) a programming assignment (implementing the discussed component), and (2) a short answer component which will test your understanding of lecture materials and weekly readings (if any). Programming and short answers will be due the following week (one week after they are assigned).

You can access each module by navigating to **Course Modules** on the course menu. A module may have several sections, and you are encouraged to preview all sections of the module before starting. You should regularly check the **Calendar** and **Announcements** for assignment due dates.

The second five weeks of the course will focus on advanced topics in language models which are used to build applications on top of the core technology discussed during the first part of the course. This portion of the course will be project-based, and will begin with students writing a brief project proposal. During these remaining weeks, students will develop a language model variation or research experiment which will be due the final week of the course (both the code and a final write-up). During this time, weekly assignments will consist only of the short answer responses, as the majority of time outside of class should be spent working on the final project.

This course is focused on building hands-on experience with programming LLMs and understanding how they work. The assignments will reflect this goal. There is no midterm or final exam.

If the course is later given in fall or spring, two additional weeks will be added during which students present their final projects to the class. The final project would, in that case, include a presentation requirement and grade.



## Textbook

There is no required textbook for this course. Readings, when assigned, will be from research publications or web content that are publicly available.

## Required Software

### Python + PyTorch

This course will make use of PyTorch (a deep learning library for python) for implementing components of language models, and assumes prior familiarity with using PyTorch for deep learning.

PyTorch is available for free from the PyTorch website: <https://pytorch.org/get-started/locally/>

We will use a few other publicly available python packages which will also be free. If these packages are not extremely standard for general python users, we will introduce them during the course.

Specific versions of python and pytorch will be mentioned during the first week of class to make sure everyone is using similar software.

## Technical Requirements

You should refer to **Help & Support** on the course menu for a general listing of all the course technical requirements.

## Student Coursework Requirements

It is expected that each module will take approximately 8–10 hours per week to complete. The weekly programming assignments will take 6-8 hours per week, with readings taking an additional 1-2 hours. In the latter half of the course (final projects), expect to spend about 4-6 hours on programming each week and 2-4 hours on your final report.

This course will consist of the following basic student requirements:

### Readings and Participation (15% of Final Grade Calculation)

You are responsible for carefully reading any assigned material and responding to short answer questions each week, which will be based on the readings and lecture material. Your responses will be graded for accuracy, critical thinking, and clarity.

Readings and participation is graded as follows:

- 100–90 = A— Short answer responses include accurate references to lecture or reading material, and show personal insight or thought into the mechanisms discussed, including an understanding of how the material fits into the larger LLM paradigm. Writing is clear and concise.
- 89–80 = B— Short answer responses contain mostly accurate references to lecture or reading material, and show understanding of the mechanisms discussed, perhaps only in isolation. Writing is generally easy to follow.
- 79–70 = C— Short answer responses contain statements of mixed accuracy or few factual references, and may only present a partial understanding of the given material. Writing is cumbersome or only partially organized.



- $<70 = F$ — Short answer responses do not adequately reflect an understanding of the material or the student having spent time with the material. Writing is hard to understand or lacking in structure and content.

## Programming Assignments (50% of Final Grade Calculation)

The majority of coursework will consist of implementing components of LLMs in python / pytorch. Program interfaces will be provided (i.e. empty class or method definitions) which should be used to structure your programs. Assignments will be submitted as .py files, and graded for code clarity and functionality (code should be both readable (clarity) and should execute correctly (functionality)).

### Clarity (30%):

Your code should be well-organized and include helpful comments where appropriate. There is no specific formatting standard required or a need to include unit tests, but as this is a 700-level course there is an expectation of quality code. Clarity grading guidelines include:

- 100–90 = A— Code is well formatted and easy to understand. Logical variable and method names are used, and common routines are broken into separate classes or methods where appropriate. Subtleties or complex components are clearly explained with comments.
- 89–80 = B— Code is reasonably formatted and relatively easy to decipher. Comments are generally helpful and are included at critical points of potential confusion. Some common routines are separated into their own classes or methods.
- 79–70=C— Code takes some effort to understand and comments are either infrequent or not helpful. Code routines are somewhat bloated or not well organized. Variable naming is poor.
- $<70=F$ — Code is very hard to understand and takes considerable effort to trace through. Code organization is non-existent or counterproductive. Comments are either not present or not helpful.

### Functionality (70%):

Submitted programs will be tested against a reference implementation and be expected to produce similar results. Any edge cases or data formatting specifics will be made clear in the assignment instructions. You are encouraged to create your own test cases to validate that your code runs correctly.

Submissions will be expected to run without errors and in a reasonable amount of time\*. If your submission fails to run you will receive zero credit for functionality.

The quantitative score for functionality will be taken directly from the number of test cases that are passed. Specific details will accompany each assignment as to the type and weight of test cases, if not uniform.

\* *reasonable amount of time*: An amount of time such that the practical grading of assignments is not impeded. There is no need to prove a specific theoretical runtime. Runtime will generally not be a significant concern unless your program gets stuck in an infinite loop or is orders of magnitude slower than the reference implementation.

## Final Project (35% of Final Grade Calculation)

A course project will be assigned several weeks into the course. In lieu of weekly programming assignments, the last few weeks of the course should be spent on the final project. Weekly readings with short answers will continue.

The final project will consist of making a variation to the GPT transformer architecture, training loop, or inference loop, and testing how this impacts the model. You may invent your own variation of interest or implement



something from existing literature. You should test your variation through experiments that show how your variation relates with other hyperparameters (i.e. what is the relationship between your design choice (and any parameters therein), with sequence length, training time, or model size, etc?). Specific examples will be given with the assignment. If you are re-implementing something from literature, seek to do something different than simply reproducing their results as well.

Note: You do not need to *improve* the existing GPT architecture (which has been optimized by the research community). You are simply tasked with understanding a specific design choice of GPT models and how it influences overall performance. A null outcome will not be penalized if it is reasonably motivated.

You may form groups of up to 3 students if you wish, but this is not required (can be individual). Groups can submit together, but will be held to a higher standard than individuals.

The final project will have the following components, which are each graded separately:

**1. Project Proposal (20%):** A written proposal including your intended variation of the GPT model, experiments to evaluate, and hypothesis as to the impact of the variation. Include all team member's names. The proposal must show critical thinking as to the variation introduced, why it may be significant, and how it will be evaluated.

The proposal is also an opportunity to get feedback on your ideas before you begin. You will be expected to incorporate proposal feedback into your final submission.

**2. Programming Component (40%):** A working implementation of your variation and code to reproduce your experiments, with instructions to run them. This will be graded similarly to other programming assignments (clarity, functionality, ease of use, etc). Your results should be reproducible by running your provided code. Specific requirements will be given when the course project is assigned.

**3. Final Report (40%):** A multi-page report detailing the typical workings of the GPT model, your proposed change and its motivation, your experiments, the results of your experiments, and conclusions that can be drawn from these results. This will follow a typical research paper / lab report structure as you may have seen in previous courses. Specific requirements (sections and content) will be given when the course project is assigned.

**Final Project is graded as follows:**

- 100–90 = A— Student or team proposes and executes a well motivated variant of the GPT model and thoroughly tests how it impacts the use of the model. Code is clean and results are reproducible. Final report is well written and clearly states the reasoning behind the project in the context of other course material, the design of the experiment, the nature of the results, and the potential consequences of the results. Feedback from the initial proposal is clearly taken into account.
- 89–80 = B— Student or team proposes and executes a variant of the GPT model and tests how it impacts the use of the model along some axes, but could be more thorough. Code is usable and results are mostly reproducible. Final report is well written and clearly states the reasoning behind the project, the design of the experiment, the nature of the results, and the potential consequences of the results. Some details are left unclear or are only reflected in the code or only stated in the paper. Feedback from the initial proposal has been mostly utilized.
- 79–70 = C— Student or team proposes and attempts to execute a variant of the GPT model and tests how it impacts one axis of the model's use or how it relates to a narrow subset of other parameters. Code is usable and reproducible with some effort or usable only under some conditions (i.e. missing edge cases). Final report contains appropriate sections but is lacking in detail and requires cross-referencing with code. Feedback from the initial proposal has been minimally considered.



- <70 = F— Student or team proposes a variant of the GPT model but it is not well-implemented or not well tested. Code requires significant effort to run or does not run. Final report does not adequately explain the project, the results, or how they link to other course content. Feedback from the initial proposal has been ignored.

## Grading

Assignments are due according to the dates posted in your Canvas course site. You may check these due dates in the Course Calendar or the Assignments in the corresponding modules. I will post grades one week after assignment due dates.

We generally do not directly grade spelling and grammar. However, egregious violations of the rules of the English language will be noted without comment. Consistently poor performance in either spelling or grammar is taken as an indication of poor written communication ability that may detract from your grade.

A grade of A indicates achievement of consistent excellence and distinction throughout the course—that is, conspicuous excellence in all aspects of assignments and the final project.

A grade of B indicates work that meets all course requirements on a level appropriate for graduate academic work.

EP uses a +/- grading system (see "Grading System", *Graduate Programs* catalog, p. 10).

100-98 = A+  
 97-94 = A  
 93-90 = A–  
 89-87 = B+  
 86-83 = B  
 82-80 = B–  
 79-77 = C+  
 76-73 = C  
 72-70 = C–  
 69-67 = D+  
 66-63 = D  
 <63 = F

Final grades will be determined by the following weighting:

Item	% of Grade
Readings and Participation	15%
Programming Assignments	50%
Course Project	35%

## Help & Support

For instructional technology support, you should refer to **Help & Support** on the course menu in your Blackboard course site. For student academic and wellness support please visit [EP Student Services](#).



## Policies and Guidelines

### External Code Resources

The subject of this course, building a language model in pytorch, is a popular one. There are many tutorials, blog posts, videos, and code repositories online which will be very similar to the programming assignments in this course. Note that these vary widely in quality. The most important thing to me is that students understand the material, so I will not (and practically could not) prevent students from accessing this content. In fact, I encourage it if it is used in a beneficial way that increases understanding. My guidelines would be:

- Always try programming assignments without looking at external code first. Referencing diagrams, equations, and class notes is perfectly fine.
- If needed, you may use an external implementation as a reference only, for occasional implementation details only:
  - Very clearly cite a reference in your code comments and explain how you used it
    - i.e. "I was unable to figure out how to implement <this specific equation> so I referenced <these lines from this resource> and found that <I was doing X wrong, how to write out Y equation, I had Z bug, etc>
  - Write your own code, do not copy many lines of code verbatim
  - Clearly comment in your own words what the code is doing to show your understanding
  - Clearly comment what you gained from the reference
- Do NOT copy code directly or plagiarize code. Use resources in a constructive way and be honest about your use. See the "Academic Integrity" section below.
- If you are very lost and feel that the only way to complete an assignment would be to excessively follow a third-party resource, please shoot me an email and we can discuss together.

### Generative AI

Do not use generative AI to author anything your turn in. Yes, it would be very ironic to do so for this course. Please do not do this.

You can of course use LLMs to help you understand how LLMs work (i.e. probe a model and observe the response). We will even be doing this in lectures. However, there is a clear line between observing LLMs and having LLMs do your work for you.

### Special Considerations for Due Dates

If you are unable to meet a specific deadline, due to either the demands of life or because you are falling behind in the course, please reach out to me. I am happy to grant extensions where appropriate and point you to additional resources. I have regular office hours and check my email frequently. However, if extensions or exceptions become a regular practice, we may need to reconsider your enrollment in the course.

## Academic Integrity

### Academic Misconduct Policy

All students are required to read, know, and comply with the [Johns Hopkins University Krieger School of Arts and Sciences \(KSAS\) / Whiting School of Engineering \(WSE\) Procedures for Handling Allegations of Misconduct by Full-Time and Part-Time Graduate Students](#).

This policy prohibits academic misconduct, including but not limited to the following: cheating or facilitating cheating; plagiarism; reuse of assignments; unauthorized collaboration; alteration of graded assignments; and unfair competition. You may request a paper copy of this policy at this by contacting [jhep@jhu.edu](mailto:jhep@jhu.edu).





## Policy on Disability Services

Johns Hopkins University (JHU) is committed to creating a welcoming and inclusive environment for students, faculty, staff and visitors with disabilities. The University does not discriminate on the basis of race, color, sex, religion, sexual orientation, national or ethnic origin, age, disability or veteran status in any student program or activity, or with regard to admission or employment. JHU works to ensure that students, employees and visitors with disabilities have equal access to university programs, facilities, technology and websites.

Under Section 504 of the Rehabilitation Act of 1973, the Americans with Disabilities Act (ADA) of 1990 and the ADA Amendments Act of 2008, a person is considered to have a disability if c (1) he or she has a physical or mental impairment that substantially limits one or more major life activities (such as hearing, seeing, speaking, breathing, performing manual tasks, walking, caring for oneself, learning, or concentrating); (2) has a record of having such an impairment; or (3) is regarded as having such an impairment class. The University provides reasonable and appropriate accommodations to students and employees with disabilities. In most cases, JHU will require documentation of the disability and the need for the specific requested accommodation.


The Disability Services program within the Office of Institutional Equity oversees the coordination of reasonable accommodations for students and employees with disabilities, and serves as the central point of contact for information on physical and programmatic access at the University. More information on this policy may be found at the [Disabilities Services website](#) or by contacting (410) 516-8075.

### Disability Services

Johns Hopkins Engineering for Professionals is committed to providing reasonable and appropriate accommodations to students with disabilities.

Students requiring accommodations are encouraged to contact Disability Services at least four weeks before the start of the academic term or as soon as possible. Although requests can be made at any time, students should understand that there may be a delay of up to two weeks for implementation depending on the nature of the accommodations requested.

### Requesting Accommodation

New students must submit a [Disability Services Graduate Registration Form](#)  along with supporting documentation from a qualified diagnostician that:

- Identifies the type of disability
- Describes the current level of functioning in an academic setting
- Lists recommended accommodations

Questions about disability resources and requests for accommodation at Johns Hopkins Engineering for Professionals should be directed to:

EP Disability Services

Phone: 410-516-2306

Fax: 410-579-8049

E-mail: [ep-disability-svcs@jhu.edu](mailto:ep-disability-svcs@jhu.edu) 

