The key learning goals of this homework are that you are able to:

- Train and deploy a machine learning model using microservices.
- Decide whether and when to use microservices, understanding benefits and limitations.
- Write an effective design document.
- Describe the architecture of a system using a suitable notation, using architectural views to represent software along different perspectives.
- Assess and choose between alternative architectural decisions.
- Engage meaningfully with questions of engineering ethics.

Project context and Tasks

You and your team are hard at work on the new graduate admissions system for CMU. During your most recent sprint planning meeting, your CMU technical contact bursts into the room, shouting, "Machine learning! Microservices!" When pressed, your newly buzzword-enamored boss explains: the population of applicants is too large, and increasing annually. Admissions is taking too much time and effort. Fortunately, CMU has been collecting student performance data. They continue: "Can't your team make use of all that fancy new machine learning to help predict which applicants are more likely to succeed, helping the admissions process along? And since we're checking off buzzwords, can't you implement it as a microservice?"

You are correctly skeptical about the effort required to pivot the product in this new direction, but fortunately the somewhat disconnected microservice demand gives you a way to explore the idea without requiring a major up front change in the existing codebase. Instead, you'll (1) develop this potential feature in isolation by training a machine learning model that predicts potential applicant success and deploying it as a microservice, and then (2) reason about, document, and choose between alternative designs that would allow you to integrate it into the existing codebase. Finally, you will (3) individually consider the ethical implications of the feature, and what questions you truly need to ask in QA to satisfy possible ethical concerns.

Starter code, data, and repository

Create a new team repository by following this link for your team, which will clone starter data and code:

https://classroom.github.com/g/PRiiOcvl

There is an example flask-based app, with associated docker file, in the dockerfile/apps folder. Use this as a template to build and deploy the microservice. You may wish to use [this article]{.underline} and [this repository]{.underline} as resources on building and saving a scikit-learn model, and then deploying it using flask.

This starter code uses arguments from the query string (specifically age, health, and absences) to query the model. This can/should be expanded or replaced as a part of your implementation (note that the API also accepts JSON requests).

Note that the default flask service in the starter code should be run, and is accessible, on port 5000. The following commands should work to build the docker container and confirm that the service is running:

> docker build -t ml:latest .
>
> docker run -d -p 5000:5000 ml
>
> curl http://localhost:5000/predict

Deadlines and deliverables

This homework has four (4) deadlines and four (4) deliverables. The first deadline (**Tuesday, Oct 27th**) is a **checkpoint** to make sure you are succeeding at building your microservice-based ML model. The second deadline (**Tuesday, Nov 3rd**) is for the **complete trained model deployable as a microservice.** The third deadline (**Thursday, Nov 5th**) is for **a design document explaining how you would integrate the new feature into your existing codebase**. The fourth deadline (**Tuesday, Nov 10th**) is for **one (1) *individual* document making an ethical argument about the feature.**

Part A: Development checkpoint - Group (due Tuesday, Oct 27th) - 5 points (~5%)

To make sure that we can help you with any challenges that come up, we have a technical deployment checkpoint.

For this checkpoint, you should simply use the default model included in the sample code, and ensure you can build and deploy the default code using flask and Docker. Because this is just a checkpoint, the expectation is that you will hit this checkpoint, and then continue to work. Therefore, when you have completed this checkpoint, you should tag a version of your code as a release for version 0.1. You can find instructions as to how to do this here:[https://help.github.com/en/articles/creating-releases]{.underline}

Then, submit this tagged version to the associated Gradescope assignment.

Part B: Microservice ML feature implementation and code-level documentation – Group (due Tuesday, Nov 3) – 55 points (~35%)

We provide you with a starter repository (see above) with a (synthetic) historical dataset (as a csv file) of student performance (this is an augmented version of the dataset [found here]{.underline}), and a (bad) model trained over it. Start by exploring the data, to understand what information is available. Student performance is described by their grades (G1, G2, and G3), where G3 is the final year grade, the target performance metric. Grades are on a scale of 0-20; 20 is the best. CMU considers a high-quality student one with a G3 grade of 15 or higher.

The existing model in the repository is very simple. Once you have familiarized yourself with the dataset, explore this model using whatever means you think are appropriate and determine in which ways you don't believe it to be suitable. Based on what you learn in this exploratory step, you should train a new model and build a deployable microservice.

You should evaluate your new model, and should explain how this model (overall) performs better than the baseline model, and any ways in which your model performs worse than the baseline model.

Plan some testing, perform that testing on your microservice, and document it. At least some of this testing should be automated.

The deliverable for this task is a documented technical artifact; submit a snapshot of your github repository to the Gradescope assignment. This should include:

(1) a trained model that is deployable as a microservice; the microservice must take as input a query request with student data and return a prediction about that student's graduate school success. This is the API for your microservice. It should also include

(2) a README that clearly and concisely explains

(a) Clear documentation for your API. (How it should be called, what data it expects, any pre-conditions for the service, and how to understand the output).

(b) A description of which features you used in training your model, and how your retrained the model performs better than the baseline model

(c) deployment instructions; and

(d) an explanation and justification of the testing you have done on it.

Part C: Design document – Group (due Thursday, Nov 5th) – 70 points (~40 %)

Once you have implemented the feature in isolation, your next (group) task is to document it and argue for how it should be integrated into the design of the existing system. Effectively, your technical contact's demand for microservices makes it straightforward to develop this feature in isolation, but it raises questions about how it should be integrated into your existing (relatively monolithic) Django-based web app. You have at least several options:

1. Rewrite the web app using a different architecture (re-architect the overall Django app in some other way, cf the Twitter case study)
2. Rewrite the feature to fit more naturally into the Django-imposed architecture.
3. Mash a microservice into the Django architecture using some other mechanism.
4. ????????

Your deliverable for this part is a design document describing the feature and how it should be integrated into the system. Submit this to the assignment on Gradescope. (NOTE: you will not have to actually implement this decision, for this or any subsequent homework in this class. You might consider implementation effort as part of your tradeoff analysis, but you should not let speculation about your workload in this class influence your decision.)

The design document must (1) concisely describe the feature, (2) make and justify a decision on how the implementation should be done, including considered alternatives and tradeoffs, and using/referencing suitable diagrams where relevant to support your argument:

1. Feature description. This should be concise (\< 0.5 pages).
2. Decision and architectural justification for how to integrate the feature into the system. This must include at least:

   a. A **concise, prioritized list of the overall *quality* requirements** you considered in arguing for the integration of the feature into the system (\<0.5 pages, soft limit). You may consider and reference the requirements you elicited during HW3 as appropriate. Rank them in decreasing order of importance. This allows readers to quickly understand what requirements you were designing for.

   b. **A design decision with a textual justification of your design.** (\~3 pages, soft limit, including alternatives and tradeoffs, below) Outline and argue for a particular implementation choice as to how the feature should be integrated into the overall system architecturally. Justify your design decisions, including why your design is adequate for the quality attributes important to this system. The justification will typically take positions regarding questions such as: Which requirements are affected by the alternatives? Which system qualities are promoted or inhibited? What styles or tactics were considered? What are the tradeoffs between the alternative solutions? Which one is better in this context and why? What assumptions did you make in your design? A good justification will refer back to the architecture documentation to substantiate the argument.

   c. **Considered alternatives and tradeoffs.** The design decision and justification *must* include description of considered alternatives, the tradeoffs they present, and why they were rejected. Tradeoffs must involve (but are not limited to) quality attributes that will be affected by the alternative. Justify such arguments with reference to appropriate diagrams (this provides traceability) and concrete examples, as appropriate.

On diagrams. The design document must include both diagrams and text. Diagrams should involve suitable architectural views; must include a legend; and should involve appropriate levels of abstraction for the components in the diagram. If necessary, use color/shape/text to differentiate between types of components and connectors. You may find it appropriate to merge more than one view into a single diagram. If you do this, you must be explicit about what views you are merging, and why. Otherwise, diagrams should clearly represent legitimate architectural views. Make sure that multiple views of the architecture are consistent with each other and the links are clear; if necessary provide a mapping in additional text.

Documenting Software Architectures: Views and Beyond, Second Edition is a useful book on creating architecture documentation. It is available (for free) as an e-book from the CMU library web site. If you use it, treat it as reference material; do not plan to read major parts of it.

Drawing diagrams is easier with the right software. Consider tools like draw.io (free, online, and collaborative), Dia, OmniGraffle, MS Visio, or even just the drawing editor of Google Docs. Google Slides will also likely work for this purpose. Pictures of whiteboard drawings are also acceptable, if *clearly* readable.

Additional Hints:

- The design document task is easy to underestimate both in terms of time needed and in terms of difficulty designing meaningful and useful descriptions. While it is easy to create a superficial solution, a good solution will likely require significant thinking, discussion, and iteration. Feel free to seek feedback from the course staff on your solution before submission.
- It may take several iterations to get your architectural views right. Appoint someone to track the accuracy and completeness of architectural representations throughout this assignment. Do not just divide up the views among your team members and assume they show everything needed. You only need to submit the final designs/documents, not intermediate steps on the process of getting there.
- As additional reference material, *Software Architecture in Practice, Third Edition* is a book on software architecture that is available (for free) as an e-book from the CMU library web site. You may wish to review appropriate sections within *Part Two* to help find appropriate *tactics*, techniques you can use in your design to promote particular quality attributes. Note that the book is *not* a reading assignment and you should not try to read it thoroughly. Instead, *use it as reference material* and select particular bits of advice that are relevant to your situation.

Part D: Ethical Discussion – Individual (due Thursday, Nov 10th) – 30 points (~20%)

Note: *this is not an open-ended reflection document like those we have requested in previous assignments. You must engage with the ethical questions at hand. *

For this deliverable, we will ask you to think about potential ethical concerns with the technology that you are proposing to implement. You should consider the ethical implications from the perspective of all potential stakeholders. You might find it helpful to consider the questions we considered in class:

1) Does the software respect the humanity of the users?

2) Does my software amplify positive behavior, or negative behavior for both users and society at large

3) Could the software's quality impact the humanity of others?

You should generate a list of potential ethical concerns for this system you are considering implementing.

For each concern, you should assess the risk of that concern (Remember risk is how likely it is to happen, and how bad would the outcomes be).

After assessing the risk, you should consider ways to address the ethical concern. For each concern, you should consider potential actions that could help address this concern, and safeguard against potential problems.

The deliverable for this task is a document (soft limit 3 pages), submitted to Gradescope that:

a) Reasons about ethical concerns when implementing an ML feature to help decide which students to accept to CMU.

b) An assessment of the risk for each ethical concern

c) A discussion of how to address each ethical concern.

d) A criteria that can be used to evaluate if the intervention described in part (c) is successful.

Grading

You will be graded as a team, with an individual component. This homework is worth 160 points. We will grade you based on the learning goals listed above.

To receive full credit for the checkpoint, we expect:

- That you have tagged a version in your github repo as a 0.1 release, and that this version will build a docker container with a trained model in Flask.

To receive full credit for the technical artifacts, we expect:

- A working artifact that outperforms the baseline model
- A discussion of how the new model was evaluated, and some evidence that it outperforms the baseline model
- A test plan that describes how the microservice was tested. This test plan should include at least some automated tests.

To receive full credit on the design document, we expect:

- The document to be dated.
- A concise, accurate feature description.
- A list of the quality requirements you considered, ranked in decreasing order of importance.
- Appropriate use of at least two diagrams to support your design argument.
- A clear decision with an accurate description of how you think the feature should be integrated into the current system (diagrams and text). Someone else should be able to read this description and understand where to start, implementation-wise. Do not say only "If X, then I'd choose design #1; otherwise, I'd choose design #2." In the real world, decisions must be made on the basis of incomplete information, with an understanding that sometimes the wrong decision gets made. You can say what information you wish you had in order to better-inform your decision.
- A good, substantive argument in favor of the design choice you made. This must include explicit description and consideration of alternatives, the substantiated (connected to quality attributes and other concerns) tradeoffs they afford, and reasons that you decided to reject them. These should be supported with technical arguments.
- A thorough discussion of the tradeoffs among the options. If there are any situations in which your choice was the wrong one, what would those situations look like, and how would you know (after the fact) that this was the case?

To receive full credit on the individual ethical reflection, we expect:

- A discussion of ethical concerns that considers a wide variety of stakeholders
- That you can reason about potential ethical concerns before the emerge
- Thoughtful consideration of what it would take to address these concerns