Project Phase 1

A CLI for Trustworthy Pre-Trained Model Re-Use

Assignment Goal

This assignment will help you learn to work as a team on a small software engineering project. It is also intended to expose you to the benefits and risks of reusing open-source software and machine learning models.

Relevant Course Outcomes

A student who successfully completes this assignment will have demonstrated the ability to

- Outcome I:
  - Identify and follow an appropriate software engineering process for this context.
- Outcome II:
  - Convert requirements into project specifications.
  - Design the software project based on two UML diagrams.
  - Implement the project.
  - Validate the project.
  - Consider aspects of software re-use, including security risks.
- Outcome III:
  - Experience social aspects of software engineering (communication, teamwork).

Resources

The following resources and links will help you understand and complete this assignment. Some additional resources and clarifications are sprinkled throughout as footnotes.

- **UML diagrams**
  - IEEE 1016-2009: IEEE Standard for Information Technology--Systems Design--Software Design Descriptions
  - UML per Wikipedia (many helpful links)
- **REST APIs (if you choose to use them)**
  - Fielding's 2000 dissertation: You can start at Chapter 5 ("REST"), but the whole thing is eminently readable and edifying.
  - 20 years later, brief commentary on what Fielding meant vs. what REST means in practice (and conjectures about why).
  - GitHub's REST API documentation.

    - Prof. Davis wrote a paper with 2 Purdue undergrads involving REST APIs, you might enjoy: https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1179&context=ecepubs
- **GraphQL APIs (if you choose to use them)**
    - The GraphQL foundation has a tutorial.
    - GitHub introduction and API docs.
    - Prof. Davis wrote some papers with IBM involving GraphQL, search the website for "GraphQL": https://davisjam.github.io/publications/
- **Hugging Face, the leading open-source model ecosystem**
    - "Getting Started With Hugging Face in 10 Minutes" ~ by Vladislav Guzey (Link)
    - Hugging Face official Wiki: https://huggingface.co/docs/transformers/en/quicktour
    - Hugging Face Course: https://huggingface.co/learn/llm-course/chapter1/1
    - Hugging Face API how to guides: https://huggingface.co/docs/huggingface_hub/guides/overview
- Prof. Davis and others have written about engineering considerations for model sharing, including reuse processes, security, and reproducibility concerns. You should read at least the abstracts of these papers. These issues should remind you why your "manager" at ACME Corp. might be nervous about fetching untrusted models from the Hub.
    - An Empirical Study of Pre-Trained Model Reuse in the Hugging Face Deep Learning Model Registry -- https://arxiv.org/pdf/2303.02552
    - PickleBall: Secure Deserialization of Pickle-based Machine Learning Models https://davisjam.github.io//files/publications/KellasChristouJiangLiSimonDavidKemerlisDavisYang-PickleBall-CCS2025.pdf
    - ModelGo: A Practical Tool for Machine Learning License Analysis – https://dl.acm.org/doi/pdf/10.1145/3589334.3645520
    - Backdoor Learning: A Survey – https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9802938

- **Project management using GitHub Projects**
    - Project management
    - Secret management
- **Software quality metrics**
    - Paper: Curating GitHub for engineered software projects.
    - Google's Scorecard project (introduction, repo).
- **Monetizing web services: selling "self-hosted" as a profit model**
    - Definition of self-hosting
    - Example: GitLab
- **Software licensing**
    - Wikipedia
    - Misc. article
    - This article is mostly about variations in commercial licensing. We thought it was interesting. It will not help you in the project, but it will make you a better engineer.
- **Postmortems**
    - Postmortems at Google

    - Postmortems at Amazon
- **Messages**
    - Blog: Writing user-friendly error messages
    - Blog: Writing meaningful commit messages

Assignment

Introduction

Your team is a subcontractor for ACME Corporation, which operates the ACME Web Service. They have recently started developing internal AI/ML services to broaden their developer offerings. So far, some of their prototype AI models have shown incredible promise. They have been using Python to build their models using PyTorch.

Based on the success so far, ACME Corporation's software architects are considering bringing up new AI-based services. They plan to offer both hardware products with AI as well as large-scale cloud-based AI services. Your team provides **infrastructure services** for ACME Corporation,[1] and you are being asked to make it easy for the service teams to get started. These service teams say they want a catalogue of the available AI/ML models that they can integrate into their services, and they are interested in information such as the datasets used to train a model and the model's performance on relevant benchmarks. The team requires a machine learning model to be provided along with its dataset and codebase, packaged in an easy-to-use manner.

You have been looking into Hugging Face, the model ecosystem for AI/ML, and are excited to see so many pre-trained models (over 1.5 million!). Your team's contact at ACME Corporation, Sarah, is open to reusing these models, but she is concerned about a few quality aspects:

- She knows open-source documentation can be sparse and wants to make sure it is relatively easy for their engineers to re-use these models ("low ramp-up time").
- She worries that open-source models might be held to a low standard of quality.
- She is worried that open-source models might not document the dataset(s) they have been trained on, and might be lacking example scripts demonstrating how to train/test the model.
- She wants to make sure that maintainers will be responsive to fix any bugs that are blocking ACME's teams.
- She said she might provide further specifications and expectations later, so your design should be able to accommodate adding new aspects.

In addition to Sarah's concerns, ACME Corporation currently offers its web service product directly via a REST API. However, she told you that in the three-year roadmap, they are exploring a licensed version of the web service that its customers can deploy internally: **self-hosted ACME**. Some considerations:

[1] ACME Corp. maintains an in-house engineering staff that establishes requirements and architectures for the software infrastructure on which they rely, but the actual design and implementation of this software is contracted out.

- In initial conversations, their prospective customers say that it will be important for self-hosted ACME to be open source so that they can tailor it to their needs.
- ACME Corporation uses the GNU Lesser General Public License v2.1 for all open-source software.
- Any models that ACME Corporation relies on could then be distributed as part of this product. Therefore, any open-source model's licenses that ACME Corporation's service engineers use must be compatible with the LGPLv2.1 license. **You may suppose that** the license description is given in the model README, under a Markdown heading "License", as in this example: https://huggingface.co/google/gemma-3-270m/blob/main/README.md?code=true

Sarah has asked your contracting team to prepare a tool to help the ACME service engineering teams choose pre-trained models wisely. She suggested that you start with a command-line interface (Project Phase 1), get feedback, and then move to a web service (Project Phase 2). She says it would be nice if your tool is "not super slow" and would appreciate some performance measurements. In addition, she prepared an initial specification. See the following for details.

Sarah's initial project specification

System input

- Should support input from command line arguments.

System implementation

- Should be majority Python
  - All code must include clear and accurate type annotations to ensure strong typing, as required by Sarah.
  - You can run these tools to ensure your codebase follows proper guidelines
    - Flake8 – Linting & style enforcement (PEP 8, code quality checks) https://flake8.pycqa.org/
    - isort – Automatically sorts and checks Python imports https://pycqa.github.io/isort/
    - mypy – Static type checker for Python, enforces type annotations https://mypy-lang.org/
  - Further Reading for Python Typing:
    - Python Typing — Official Docs: https://docs.python.org/3/library/typing.html
    - PEP 484 – Type Hints: https://peps.python.org/pep-0484/
    - PEP 561 – Distributing and Packaging Type Information: https://peps.python.org/pep-0561/

System output

- Should print all output to stdout (though this output mode might change in the future, so design accordingly).
- Each model should be accompanied by its overall score, as well as its sub-scores for "size", "license", "ramp up time", "bus factor", "available dataset and code score", "dataset quality", "code quality", "performance claims"
- Each model can also have a linked database and code objects.

Sarah's other requirements

Sarah wants to know how long it will take your system to respond. Please select and justify some representative input (i.e. example models/datasets) to show the latency of ingestions as models vary, e.g. in size.

For better latency, Sarah wants each metric to be calculated in parallel, although she acknowledges that you may wish to consider the number of available cores when deciding on the level of parallelism.

Auto-grader API (To make the course staff's lives easier!)

We will auto-grade the I/O behaviors of the project. To this end:

- There should be an executable file in the root directory of your project. This file should be named "run".
    - *NB: When we say that this file should be executable, that means its permissions are set to executable. That does not mean it must be a compiled program, but it could be. You can write this executable file in whatever programming language you want. Just run "chmod +x run" on it to set its permissions to executable.*
- To facilitate auto-grading, this file called "run" should have the following CLI when executed on a Linux machine:
    - "./run install"
        - Installs any dependencies in userland (e.g. *pip install --user*).
        - Should exit 0 on success, non-zero on failure
    - "./run URL_FILE", where URL_FILE is the absolute location of a file consisting of an ASCII-encoded newline-delimited set of URLs.
        - These URLs may be of the following three categories:
            - Model URL (e.g. https://huggingface.co/google/gemma-3-270m/tree/main)
            - Dataset URL: (e.g. https://huggingface.co/datasets/xlangai/AgentNet)
            - Code URL: (e.g. https://github.com/SkyworkAI/Matrix-Game)
        - This invocation should produce for the model URLs only. (Assume that if the dataset and code are linked to the model, it would appear before the model URL.) Each output must include **exact** fields (case sensitive) ***as mentioned in*** Table 1 ***below***
            - Each score should be in the range [0,1] where 0 indicates total failure and 1 indicates perfection. The specific operationalizations are up to you; you must design and justify them in your report.
            - Latency values should reflect the time to calculate that component of the net score. Report values in milliseconds and round results to zero decimal places (i.e., to the nearest millisecond).
            - The "NetScore" should also be calculated in the range [0,1], as a weighted sum. You should choose the weights based on Sarah's priorities and explain your choice.
            - Should exit 0 on success, non-zero on failure
    - "./run test", which runs your test suite.

- The minimum requirement for this test suite is that it contain at least 20 distinct test cases and achieve at least 80% code coverage as measured by line coverage.
- The output from this invocation should be a line written to stdout of the form: "***X/Y test cases passed. Z% line coverage achieved.***"
- Should exit 0 on success, non-zero on failure

In the event of an error, your program should exit with return code 1, and print a useful error message to the console. Look at the resource on error message design for guidance.

Your software must produce a log file stored in the location named in the environment variable[2] $LOG_FILE and using the verbosity level indicated in the environment variable $LOG_LEVEL (0 means silent, 1 means informational messages, 2 means debug messages). Default log verbosity is 0.[3]

Before submitting, ensure you run your software on the ECEPROG server[4] and confirm it runs successfully according to the "auto-grader" interface. This is to ensure your software compiles and runs successfully when we try testing or auto grading your software.

The course staff will publish input/output examples for you to test with. These will be available on Brightspace.

[2] In documents like this one, environment variables are written in ALL_CAPS, use snake_case, and sometimes have a $ in front. When you read "the environment variable $GITHUB_TOKEN" it means that there is a variable defined in the environment whose name is GITHUB_TOKEN, which you should access using (in python) something like: `githubToken =os.environ["GITHUB_TOKEN"]`

[3] There are many views on what and how much to log. Some good resources are (1) this blog; and (2) this SO post.

[4] ECEGRID has retired. See https://engineering.purdue.edu/ECN/Support/KB/Docs/ECEThinlinc for the new machines.

Table 1: Required NDJSON field, their type, and their definitions

Field Name	Type	Range/Options	Notes
name	*string*	–	Model / dataset / code name
category	*string*	[MODEL, DATASET, CODE]	Category type
net_score	*float*	0 – 1	Overall quality score
net_score_latency	*int*	milliseconds	Time to compute net_score
ramp_up_time	*float*	0 – 1	Ease of ramp-up
ramp_up_time_latency	*int*	milliseconds	Time to compute ramp_up_time
bus_factor	*float*	0 – 1	Metric measuring the knowledge concentration (higher = safer)
bus_factor_latency	*int*	milliseconds	Time to compute bus_factor
performance_claims	*float*	0 – 1	Evidence of claims (benchmarks, evals)
performance_claims_latency	*int*	milliseconds	Time to compute claims
license	*float*	0 – 1	License clarity & permissiveness
license_latency	*int*	milliseconds	Time to compute license info
size_score	*object {str -> float}*	raspberry_pi	A dictionary mapping hardware types to floats (0–1), indicating model size compatibility with each device
		jetson_nano	
		desktop_pc	
		aws_server	
size_score_latency	*int*	milliseconds	Time to compute size score
dataset_and_code_score	*float*	0 – 1	If the dataset used for training and benchmarking is well documented, along with any example code.
dataset_and_code_score_latency	*int*	milliseconds	Time to compute availability score
dataset_quality	*float*	0 – 1	Dataset quality
dataset_quality_latency	*int*	milliseconds	Time to compute dataset quality
code_quality	*float*	0 – 1	Code style, maintainability
code_quality_latency	*int*	milliseconds	Time to compute code quality

Metric calculations

At least one metric must use data from the Hugging Face Hub API

Many Hugging Face models and datasets are hosted on the Hub. Your software only needs to support metric calculations for resources on the Hub (though it should still handle "other hosting" gracefully).

- Example Hugging Face API data you can use:
  - Number of downloads (`downloads` field).
  - Likes / stars (`likes` count).
  - Last modified date (for freshness).
  - Number of model files / artifacts in repo.

At least one metric must analyze the model repository without using the Hugging Face API.

- To do this, you should clone/download the model repo locally
- Then you can analyze:
  - You might want to interact with the Git metadata programmatically. If you do so:
    - You *cannot* implement analysis by "shelling out" to the git bash CLI
    - Instead, use a Git library, such as isomorphic-git (https://github.com/isomorphic-git/isomorphic-git).
  - Model files themselves:
    - Inspect `config.json`, `model_index.json`, or `README.md` for architecture, training details, datasets used.
    - Check the size of weights (`pytorch_model.bin`, `tf_model.h5`, etc.) for deployability.
    - Verify presence of tokenizer/vocabulary files.
  - Static analysis → Parse `config.json` / metadata for structure.
  - (For Extra Challenge) Dynamic analysis → Optionally, load the model with `transformers` and check:
    - Does it load without errors?
    - Are test scripts (`test.py` or sample notebooks) passing?

Source code hosting

Your team's repository should be shared publicly on GitHub.

- This will promote good practices with respect to tokens and keys. (See Resource "Secret management".)
- It will allow you to share it with future employers if you are so inclined.

- Per the academic honesty requirements of this course, and unless otherwise indicated, you are not permitted to work with other teams, compare or copy software implementations, and so on.

Project management

You must use GitHub Project Boards for progress tracking. Your weekly milestone updates should use metrics obtained from the board, possibly including screenshots.

Software re-use

You are allowed to re-use existing software to support your implementation, either as tools (e.g. VSCode; git; GitHub; TravisCI) or as components in your implementation (e.g. a module to help you parse command-line arguments). Your plan and final report should include justifications of any components you choose to re-use – how will you decide whether they are trustworthy? (discuss in the Project Plan) And how did that assessment work out in practice? (discuss in the Project Postmortem)

You are allowed to reuse code snippets from software engineering resources such as Stack Overflow. You must provide a citation (web URL is fine) to the relevant post. Also, please review Prof. Davis's general perspective on Stack Overflow. Technically, Stack Overflow snippets are themselves governed by a software license, but you are not trying to distribute your project code, and I do not think anyone would seriously pursue litigation along these lines.

You are **not** allowed to copy-paste code snippets out of an open-source project – this is a great way to expose ACME Corporation (and your future employer in the real world) to lawsuits. Any re-use of this nature must use existing module APIs and/or extend those APIs so that you can access the logic you want.

Your team **must** use one or more large language models. Here are some suggestions:

- **(Required) To analyze the README and Metadata of a Model**: You can use any free service like Amazon SageMaker, or make a Pipeline with Purdue GenAI studio API, to analyze the README or other files and compute the required metrics. (Further Reading: PTMTorrent)
- **(Required) To accelerate your implementation**: Within your IDEs, there are many pluggable implementation-type LLMs that can help you write code. Two examples are GitHub's CoPilot and Meta's Code Llama. As a student, you can access GitHub's tools through the GitHub Education program.
- **(Optional) To support your other software engineering activities**: At the time of writing, many companies offer free chatbot-style LLMs, including Claude (Anthropic) and ChatGPT 5 (OpenAI). Through Purdue, you have access to Office 365, which includes Microsoft's commercial tool Microsoft CoPilot (not to be mistaken for GitHub CoPilot). These tools can support many other phases of the software development lifecycle, such as:
    - Requirements analysis ("*ChatGPT, what does the following confusing passage from Prof. Davis's giant Word Doc mean?*")

  - Software design ("*How might these components communicate?*")
  - Validation ("*Any recommendations for automated testing of a component that does X?*")
  - Project management ("*Here are our skills. How might we divide up work?*")
  - Learning new skills and technologies ("*Can you summarize the different AWS Free Tier options for persistent data storage?*")

Your Project Plan should include a description of how you used the LLM in a responsible way. Remember that the course has a "Bring Your Own Brain" policy. If the LLM makes a mistake, then it is you who will bear the consequences. Here is some general advice from Prof. Davis on the proper use of tools and brains.

Timeline and Deliverables

The project will be completed over a 5-week period:

- Week 1: Planning and Design
- Weeks 2-4: (Inevitable re-designs, and) Implementation, Validation, Delivery
- Week 5: Postmortem

Week 1: Design and Planning

One member of your teams should submit a Project Plan Document (Word Doc or PDF) including the following. The template is available on Brightspace and is the master, but here's a summary.

- Tool selection and preparation
  - Toolset, component selection [linter? Git-hooks? CI? Testing framework? Logging library?]
  - Communication mechanism(s) [Slack? Teams? Email?]
  - Statement that GitHub tokens are obtained and a repo created
- Team contract
  - For example, your team might agree to do the work they take on, document their code, follow testing rules, follow style guide, and to communicate in advance if they cannot deliver on schedule
- Team synchronous meeting tempo and times
  - We recommend at least one (short) mid-week sync to discuss issues, and one end-of-week sync to put together your weekly reports.
- Requirements
  - A refined and organized list of requirements, based on Sarah's description and specification.
- Preliminary design
  - Metric operationalizations and net score formula
  - Diagrams to support planning. These should be drawn using LucidChart or similar. We expect at least two, imitating the purpose (if not the exact style) of these UML diagram types:
    - UML Activity Diagram to depict the activities performed by your system.
    - Simplified UML Class Diagram to depict the critical entities in the system and how they will relate to each other.

    - Explanation of the design of the "metrics" feature so that you can accommodate Sarah's projected need to add new metrics later. What logical flow and what entity structure (e.g. class hierarchy?) did you select to improve the modularity of this portion?
    - Explanation of the design of the "handle URLs" feature so that you can accommodate URLs from various categories of objects. What logical flow and what entity structure (e.g. classes, hierarchies?) did you select to improve the modularity of this portion?
- Planned milestones for weeks 2-4
    - Each milestone should list the necessary tasks, the expected owners of those tasks, the estimated time to complete it,[5] and how success will be measured.
    - Any communication requirements between tasks should be noted, e.g. "Jason and Tahani need to discuss the interface involved between task A and task B."
- Validation and Assessment plan
    - What is your plan to assess whether the delivered software satisfies Sarah's requirements? What behaviors will you check? What performance metrics (if any) will you apply?

Weeks 2-3: Complete your internal milestones

Each week, submit a report with your updated list of milestones, tasks, etc. representing completion and the actual time spent by each team member on the project.

This report should be self-contained, e.g. including the relevant information from the original plan.

If you *deviate substantially* from your timeline, consider attending one of the course staff office hours to discuss the deviation.

Week 4: Deliver the software

Submit the software itself, along with a report describing the status of the software in relation to Sarah's requirements and specifications.

- If your submission will not survive the auto-grader described above, provide explanatory notes so the course staff can score you fairly.[6]
- Provide one example of a model that you think your approach scores well on.
- No automated measurement is perfect. Provide one example of a model that you think your approach scores poorly in some regard. How could you modify your design to improve the outcome for this model?
- Provide the URL to your project repository in your report

[5] Bad news: You are bad at estimating how long things will take. Good news: With careful practice, you can get better. As a simple rule of thumb, add 50% to your team's best guess. For more reading, this blog post is chock full o' wisdom.

[6] Pro-tip: After you have a design, build an end-to-end skeleton to get the interfaces working.

Week 5: Postmortem

Deliver a project postmortem report. This report should reflect on each aspect of your Plan (from week 1) compared to your Execution. What went well? What went poorly? Where did your time estimates fail? When and why did you deviate from your Plan? For all these questions, try to answer the question "Why?"

See the resources on "Postmortems" at the beginning of this document.

Templates for all deliverables will be provided in Brightspace as they become available.

Grading rubric

Points breakdown:

- 20% Design & Planning document
- 10% Milestone documents.
- 60% Working delivery, broken down as:
  - 30% "It runs and follows the auto-grader interface above"
  - 10% "It has a reasonable-looking test suite that achieves the required coverage"
  - 10% "Per our manual inspection, the software follows reasonable-looking engineering practices, e.g., good file/class/variable names, consistent style, choice of data structures, use of patterns to isolate what is changing".[7]
- 5% Hand-off
- 5% Post-mortem.

Provided that the teammates complete the tasks they were assigned as part of the project plan, all team members will receive the same grades. If there is an issue with teamwork, please raise it with your professor as early as possible.

- Your team's milestones should allow you to observe problems with forward progress.
- For personality clashes, etc., use your judgment to determine if you want to speak with your professor.

ACME Corporation's Budget is not Bottomless

Sarah reminds you that your team members are from an independent contracting firm. She says the company is **willing to pay your team for up to 40 hours per person for this project**[8], and would rather see ***something that works – at least partially! – by the deadline.***

- Your project plan and your weekly progress updates should reflect an appropriate amount of time for the project, e.g., 6-9 hours per team member per week. If you wait until the

[7] Specific elements assessed in the grading rubric include (1) the presence of a main README and in any sub-directories to document the design – no project should leave home without them! – and (2) suitable module-level (top of file) and function-level comments documenting the implementation

[8] "40 hours per person for this project" – Since the project will run for 5 weeks, that will average out to ~8 hours per teammate per week. The postmortem week should be lighter and the earlier weeks a little heavier.

last minute, Sarah will be nervous, pull the plug on the project, and might break off future contracts with your company.

- If you begin to deviate from your planned timeline, you should submit a **revised** plan as part of your Week 3 update. That way Sarah can keep management abreast of progress and aware of any changes in the functionality that will be delivered.
- You should plan your project in such a way that you can deliver incremental value to Sarah even if you cannot complete all her requirements.
    - Recall the aircraft requirements document from the Requirements Engineering unit – one of the final chapters designated useful subcomponents that the vendor could deliver.