# WASP software engineering assignment

Livia Qian

August 2023

## 1 Research topic

I am Livia Qian and I am a second-year PhD student at KTH Royal Institute of Technology in Stockholm, Sweden. My research area is conversational AI which includes both written and spoken dialogue modeling for the purpose of creating representations that can be used to solve a wide range of tasks, e.g., emotion detection, sarcasm detection, laughter detection, turn taking, voice activity detection and next-sentence prediction.

Incorporating dialogue systems in conversational agents and traditional software has become increasingly important with the development of social robots, chatbots and live captioning. This field is slowly but surely expanding, although the current state-of-the-art in natural language processing (NLP) and similar areas focus mainly on monologues, and even when they incorporate multi-party conversations, they do not take into account nonverbal cues like prosody and pauses. To improve the quality of the use cases mentioned before, it is therefore crucial to focus on paralinguistic elements of both text and speech.

For the purpose of modeling dialogues, I usually use machine learning models that need to be trained to retain information from different types of dialogue data. In the case of spoken dialogue modeling, the training speed is usually massive and the data is extremely hard to store due to the nature of sound and the requirements imposed by the most widely used automatic speech recognition (ASR) and speech synthesis (TTS or text-to-speech) systems like wav2vec2 [2] and Whisper [6]. For written dialogue modeling, I usually use the most recent natural language processing models, e.g., GPT-2 [7], GPT-3 [3], ChatGPT [1] and LLaMA [8], which are pre-trained on large amounts of text data and should preferably be fine-tuned on big datasets.

Although many pre-trained models, code bases and datasets are free to access and usable out-of-the-box, more complex tasks demand that they be modified. In many cases, the modifications result in massive changes in the pre-existing code base. Besides, since my project is about processing conversational data -– which implies that there are at least two participants, i.e., two channels or streams of information —, there is double (triple, etc.) the amount of data that needs to be handled compared to when there is just a single speaker. These are why it is of utmost importance to use software engineering principles.

## 2 Principles and concepts relating to my research area (main lectures)

The first and probably most obvious principle when speaking of code changes is **version control**. As in classic software development, it is important to be able to keep track of changes for error detection and reversibility. Although many projects are individual projects or modularized so that every person is responsible for different parts, it is common that every member of a research group or a temporary collaboration develops and revises the same

[1] https://chat.openai.com/

pieces of code, which is why concurrency, annotations and proper documentation are important. Moreover, atomic commits should be used for additional readability, meaning that every self-contained change should correspond to a single commit and nothing more. It is also advisable to keep every module or system component on separate branches. Some of the most popular version control systems are Git [2], Concurrent Versions System [3] (CVS) and Apache Subversion [4] (SVN). I use Git in my everyday work because it has enough functionalities to cover what I need (code versioning).

Another important concept is **verification and validation**. There are many ways to verify code correctness and efficiency; the list includes methods like code review, manual tests, static source code analysis and dynamic analysis. With respect to the atomicity of the components to be tested, there is usually a distinction between unit tests, integration tests and system integration tests.

In the context of my research, the models are usually not part of a bigger product but they still consist of multiple components. *Unit tests* and assertions are placed primarily to check smaller functions or classes; common tests include checking the most common cases (e.g., dummy data that is representative of the actual data), the expected output (e.g., correct dimensions) and the edge and invalid cases (e.g., when the input is a null or infinite matrix). I usually place assertions whenever I need to test simpler code blocks (e.g., the values stored in some variables) and write unit tests for more complex functions and processes.

*Integration tests* (I&T) are conducted on software components like the model, the dataset processing and the evaluation pipeline; in these cases, when the individual components are highly intertwined, they can be replaced by simpler versions (e.g., a synthetic dataset, a fraction of the training set and a smaller version of the model in question). This is where processes like feature extraction, transformations and batch normalization can be tested.

*System integration tests* (SIT) are meant to validate the correctness and quality of the system as a whole. Here, the communication between the different modules are the main focus and the whole training and inference pipelines are investigated by e.g., checking what the training and evaluation curves look like and comparing the performance to baselines while adhering to the original hypothesis.

To save time by reducing the number of potential future bugs, *test-driven development* (TDD) may be used in cases where we know what the expected output should be. Once a unit of code is ready, we can run automatic tests that can show where and why certain cases failed. *Stress testing* is also useful when we know that resources are limited and that training the model will be compute-intensive so that it does not break halfway through. Normally, I try to do things like increase the batch size and train a smaller model on the most complex data samples.

*Static code analysis*, that is, when the checks can be done without having to run the code, is especially effective in the case of dynamically typed languages like Python since they are usually weakly typed and thus do not inherently do strict runtime checks. *Manual code analysis* is usually done by checking formatting errors, refactoring and retesting the code and isolation testing where each major component is inspected separately. Finally, we can use tools like Jenkins [5] that can automate the building, testing and deployment of a software, facilitating continuous integration (CI); it can set up the various commands and tests in order, which is a convenient way to train, search for hyperparameters and run inference.

[2] https://www.github.com

[3] https://cvs.nongnu.org

[4] https://subversion.apache.org

[5] https://www.jenkins.io

# 3 Principles and concepts relating to my research area (guest lectures)

**AI for Software Engineering** (AI4SE) is the use of artificial intelligence techniques as tools in software development. Common tools are *intelligent code completion* where keywords, variable names, function names, and the template of the documentation are autocompleted by an external software. In the traditional sense, it refers to when completions are suggested based on predefined rules and static analysis, but with new technologies like Copilot [6], bigger blocks of code can be inserted as recommended by machine learning models. Copilot has been trained on pre-existing code and it infers what should be written at a specific point. This is very useful when writing (unavoidable) boilerplate codes like training loops and function calls characteristic of specific frameworks (e.g., Huggingface [7]). Copilot is an example of a **DevBot** that automatizes a part of the code writing process. Sometimes I use it to write lines that I know have been written by many people before.

DevBot are used in other contexts as well. In many cases, we can ask chatbots to help highlight the part of a function's or command's documentation that can be relevant to us given the problem at hand. Nowadays, for example, ChatGPT can answer questions regarding certain classes and methods in a framework or even provide an outline of what should be done to set up an entire project from scratch. This can happen when, e.g., I need to set up a user study on Amazon Mechanical Turk [8] to evaluate the speech samples my model generates and I want to know how HITs (human intelligence tests) work and how they should be created, released and evaluated automatically.

# 4 Software engineering topics

## 4.1 Project management

Project management is the process of planning and assessing project activities; it consists of defining, organizing and executing the tasks of a team, as well as assigning roles and responsibilities to the members in order to maximize the success of a project. Project management is an important part of the software development pipeline irrespective of field and context. The tasks are defined according to the goal and requirements of the project, with the main resources being time, scope and budget. The customer's or end-user's needs are taken into account during specification and, if possible, continuously discussed with and revised by them. The roles are dealt according to expertise and efficiency, while the scope is adjusted to temporal, monetary and infrastructural constraints, which usually leads to a minimum viable product (MVP) in the first iteration.

There are many project management strategies, e.g., agile, extreme and continuous integration, which incorporate an even wider range of software development processes, e.g., Scrum, Lean and Waterfall. Although Waterfall is a traditional methodology used in other fields like mechanical and civil engineering where the nature of the product demands linear sequential execution, software engineering usually benefits from smaller, continuous and iterative steps.

In machine learning, human-in-the-loop is widely used to continuously update and improve the system. This means that the project exploits the combined effort of machine and human intelligence. Humans are involved in the development in the sense that they provide feedback after each iteration, either explicitly as instructed by the developer team or implicitly through the casual use of the system. Such a process requires even more complex project planning strategies as the staged release of the product is intended for a wide range of – in many cases unknown – users.

[6] https://github.com/features/copilot

[7] https://www.huggingface.co

[8] https://www.mturk.com

Software maintainability prediction is a crucial part of a project as it is about ensuring high quality by optimizing resource allocation, cost utilization and the management plan [1]. Various machine learning methods can be used to address quality and maintainability prediction: K-means clustering, support vector machines (SVM), generalized regression neural network (GRNN) and AdaBoost, among others. These are based on a range of factors are metrics, such as number of lines in the source code, depth of inheritance tree (in the case of object-oriented programming), number of tasks requiring software modification, total number of tasks, man-months for testing, maintenance time and maintenance cost. Similar problems include classifying new tasks based on type (e.g., story, bug, epic) and severity (urgent, normal, nice-to-have) with decision trees, estimating the budget of a project based on past projects with linear regression and identifying potential outages through anomaly detection [5].

I believe that there is a benefit to adding these types of predictions to issue tracking products like Jira [9]. Predictions and planning recommendations based on machine learning can improve the accuracy and complexity of key performance indicators (KPIs). Cost and time estimation can also be relevant to my PhD project, especially when I work with many people and the task allocation becomes increasingly complex. Moreover, they can possibly help with giving an indication to the expected performance of my machine learning models, measured in terms of e.g., accuracy in classification tasks and word error rate (WER) in transcription-related tasks.

### 4.2 Security and privacy

Security and privacy are crucial aspects of software engineering whenever there is a possibility for system failure and user data is involved. Privacy, in short, concerns the collection, storage and handling of sensitive or personal data, while security is the protection of the software from external attacks or internal failures, which may or may not lead to data breach. Privacy has come to the fore in recent years, especially with the introduction of the General Data Protection Regulation (GDPR) in the European Union in 2018. Security has always been a concern, especially for governmental organization that fear data leak and the unauthorized distribution of classified documents.

Although privacy and security are constantly addressed in the community, the exponential growth in size of new models and datasets leads to more and more problems [4]. Platforms for building and sharing machine learning models might be careless about how users expose their models and data to external users or might share their properties with third parties. When it comes to active black-box attacks, attackers might try to infer whether certain samples are from a specific model's training data by assuming that the model would overfit on these samples, therefore, they measure the confidence score for each input-output pair. This can be done for both discriminative and generative models, although it is harder in the latter case. The most effective way to exploit this case is to train a generative adversarial network (GAN) to detect overfitting.

Security issues can also concern the machine learning models themselves if the adversary's intention is to steal the parameters (model extraction) [4]. Model extraction is usually done by reverse engineering the parameters by querying random inputs if the architecture is known. Functionality extraction, i.e., training a knockoff model can be done via training on the input-output pairs observed.

Defense strategies include technologies like cryptography and differential privacy (DP) [4]. DP is a state-of-the-art method that protects sensitive information by adding noise to the data samples. Regularization and normalization methods help alleviate overfitting which makes it harder to infer the input-output pairs in the training set, although these approaches are not particularly robust.

The type of attacks mentioned before are yet to be studied and avoided as these problems are very real [4]. Additionally, privacy guidelines and regulations should also be introduced or revised to provide extra security. These

[9] https://www.atlassian.com/software/jira

problems are present in my area as well as the datasets I use contain information about speakers that can make them identifiable even when meta-information about the samples are anonymized. An example of information that can be exposed is the speaker's voices themselves as they are more often than not kept in their original form.

## 5 Future trends in software engineering

I believe that in the upcoming years, more and more importance will be placed on AI-based automation. When it comes to code completion, it is already a great achievement that blocks of code can be generated based on just a single prompt; I think that this will improve in the future with the incorporation of online NLP, knowledge bases and web crawlers. What I can think of is plugins that can suggest code completions based on online tutorials and official guides or write bigger chunks of the documentation. I can also see these automation tools being able to suggest datasets for different tasks and automatically clean and apply the necessary transformations on these datasets to suit our needs. In the case of speech data, I can see them apply automatic feature extraction, different types of transformations (e.g., Fourier transform) or pretrained representations depending on the task at hand. Additionally, I think that future tools will be able to define the basic structure of and provide a template for an ML project, with all the necessary scripts, configuration files and frameworks already in place.

# References


[1] Hadeel Alsolai and Marc Roper. A systematic literature review of machine learning techniques for software maintainability prediction. *Information and Software Technology*, 119:106214, 2020.

[2] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *CoRR*, abs/2006.11477, 2020.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[4] Emiliano De Cristofaro. A critical overview of privacy in machine learning. *IEEE Security & Privacy*, 19(4):19–27, 2021.

[5] Nikos Kanakaris, Nikos I Karacapilidis, and Alexis Lazanas. On the advancement of project management through a flexible integration of machine learning and operations research tools. In *ICORES*, pages 362–369, 2019.

[6] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022.

[7] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

[8] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.