# Mini-Project List (2021)

## Alex:

**Using Open Source Name: Flower**
**Project title:** Optimizing Federated Learning Hyper-Parameters in Flower

Flower is an open-source Federated Learning framework. Unlike other frameworks which mainly serve to simulate FL execution, it is intended to be usable in large-scale practical Federated Learning. Flower offers a variety of datasets, Federated Learning aggregation algorithms, and ways to configure data distribution and client selection across the FL network. Furthermore, it provides a set of standard benchmarks widely used in FL research. The intent of the mini-project would be to add an optimization module to Flower capable of tuning the hyper parameters---local/global number of epochs, client number, local/global learning rate e.t.c--- of a combination between an ML model and a global FL aggregation model. This module is intended to be built using Bayesian Optimization (potentially) in BoTorch or a similar framework and to allow for customization across different models and algorithms. Once the optimization component is built, a comparative analysis of expert-tuned vs auto-tuned performance for the benchmarks available in Flower will be performed.

## Andreea:

**Using Open Source Name: TensorFlow, PyTorch, and possibly Horovod**
**Project title: Comparative analysis of distributed data parallel training packages**

Develop a toy model and write it in both TensorFlow and PyTorch and evaluate devise some experiments to evaluate the performance of their native distributed data parallel training solutions (mirrored strategy in TensorFlow and DDP in PyTorch). This demonstrates a performance comparison between TF Mirrored Strategy and PyTorch Distributed Data Parallel. It can be extended by looking into Uber's Horovod as well, which can be added on top of the bare TensorFlow code and on top of the bare PyTorch code respectively with those results into the comparative analysis.

## Brady:

**Using Open Source Name: Spark + GraphX**
**Project title: Link Prediction with GraphX and Spark**

Link prediction is the problem of predicting the existence of a link between two entities in a network. In this project, I'm going to use the Citation Network Dataset [1] to predict future co-authorships. I will use the GraphX API to build a co-author graph and perform both unsupervised and supervised learning on it. For unsupervised learning, I will use the node similarity algorithms (Common Neighbours, Jaccard Similarity, etc) presented in [2]. For supervised learning, I will incorporate the similarity score from unsupervised learning as features and use it to train an ML model (possibly random forest classifier). I will compare the performance of both supervised and unsupervised learning in the end.

[1] Citation Network Dataset: https://www.aminer.org/citation
[2] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. J. Am. Soc. Inf. Sci. Technol. 58, 7 (May 2007), 1019–1031.

## Charlie:
**Using Open Source Name: Botorch (+SigPost)**
**Project title:  Hyperparameter Optimisation of Neural Networks**

## Conor:

### Using Open Source Name: Snorkel + Emukit
### Project title: Multi-Fidelity Supervised Learning with Snorkel

Snorkel [1, 2] is an open source tool for creating large training data sets from unlabeled data. Using this data, deep neural networks can be trained almost as accurately as models trained on large hand-labeled datasets [1]. However, although hand labeled data (high-fidelity) is expensive in comparison to the noisy (low-fidelity) labels generated by Snorkel, it is often easy to access a small amount of this high-fidelity data. I propose to modify the process for training models based on generated data to take advantage of small sets of high-fidelity data. Multi-fidelity modeling has already had success in applications to physical processes and function approximation [3, 4, 5], and I will attempt to extend this to more generic supervised learning problems. I will test a variety of different composite models on datasets that Snorkel-generated data has already been applied to such as relation extraction in text [1], using a variation of the tools in Emukit [3] for multi-fidelity emulation.

References:
[1] A. Ratner, S. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. R´e: Snorkel: Rapid Training Data Creation with Weak Supervision, VLDB, 2017.
[2] A. Ratner, B. Hancock, J. Dunnmon, R. Goldman, and C. R´e: Snorkel MeTaL: Weak Supervision for Multi-Task Learning, DEEM, 2018.
[3] Andrei Paleyes, Mark Pullin, Maren Mahsereci, Cliff McCollum, Neil D. Lawrence, Javier Gonzalez. Emulation of Physical Processes with Emukit. arXiv:2110.13293. 2021.
[4] Xuhui Meng, George Em Karniadakis. A composite neural network that learns multi-fidelity data: Application to function approximation and inverse PDE problems. Journal of Computational Physics. 2019.
[5] Xuhui Meng, Hessam Babaee, George Em Karniadakis. Multi-Fidelity Bayesian Neural Networks: Algorithms and Applications. CoRR. 2020.

## Mihai:

### Using Project name: Spark
### Project title: Recommender systems with dimensionality reduction in Apache Spark - A trade-off between runtime and accuracy

Apache Spark can be used to train a recommender system using collaborative filtering [1]. Spark also supports dimensionality reduction [2] using singular value decomposition or principal component analysis. This project will explore the trade-off between runtime and the
accuracy of collaborative filtering [3]. In the same direction I'd like to explore the use of clustering techniques [4] as a way to apply
dimensionality reduction in Spark. For example, in a movie recommender system we can cluster the M movies into K clusters and use those K clusters as features to the recommender system. A prediction for a user and a movie will be based on the predicted rating of that cluster or a weighted combination of the closest K' clusters.

[1] https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html
[2] https://spark.apache.org/docs/latest/mllib-dimensionality-reduction.html#principal-component-analysis-pca
[3] https://medium.com/@jwu2/improving-collaborative-filtering-with-dimensionality-reduction-a99d08585dab
[4] https://spark.apache.org/docs/latest/ml-clustering.html

## Samuel:

**Using Project name: SABER**
**Project title: Reproduction, Verification, and Improvements for SABER**

1. Reproduce results on my desktop computer (AMD Ryzen 7 1800X 8-core + GTX 1080 GPU)
2. Verify GPGPU data pipelining is as expected (figure 6), using NVIDIA Nsight profiler.
3. The paper notes a limitation: "the computation of window boundaries is always executed on the CPU", which limits the throughput of the GPGPU JOIN query (Figure 12c). I'd like to investigate moving this to the GPU, and seeing how it affects the results.

This will make use of my prior GPGPU+CUDA experience and require me to understand the codebase, particularly the interoperation between the GPU and CPU.

**Wanru:**

**Using Project name: Ray & Spark**
**Project title: Scaling Emerging AI Applications with Ray, the Successor to Spark**

Ray, a general-purpose cluster-computing framework that enables simulation, training, and serving for AI applications. It retains all the desirable features of Spark, and supports just-in-time, data-flow type architecture, where a task goes and all the tasks it depends on are ready and finished. Nowadays, there is a federated learning framework building on Ray to schedule the execution of client-side tasks. In case of limited resources, Ray can sequence the execution of client-side computations, thus enabling a much larger scale of experiments on common hardware. Therefore, starting with comparing Ray with Spark at the implementation level, the study is going to explore the advancement and applicability in the emerging AI applications.

**Zak:**

**Using Project name: Placeto**
**Project title: Alternative Graph Embeddings for Placeto**

Placeto [1] is a RL-based device placement technique. Placeto uses a GNN to learn node representations from a computation graph, followed by a policy network which assigns each node to a device. Beyond traditional message-passing, Placeto also encodes the set of all upstream, downstream, and unreachable nodes into each vertex's representation in order to capture higher-level information about graph structure. They found this technique crucial for achieving state of the art performance. I'd like to explore applying other embedding techniques to accomplish the same goal, namely:
- Directed Acyclic Graph Neural Networks (DAGNN) [2]
- Position-aware Graph Neural Networks (PGNN) [3]

If I achieve that, I can also look into applying methods other than REINFORCE for the policy network, such as PPO, and benchmarking the differences.

Code is available for Placeto, DAGNN, and PGNN, however Placeto's code is rough and will likely take some time to get running.

[1] [1906.08879 Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning](https://arxiv.org/abs/1906.08879)
[2] [2101.07965 Directed Acyclic Graph Neural Networks](https://arxiv.org/abs/2101.07965)
[3] [Position-aware Graph Neural Networks)(https://arxiv.org/abs/1906.04817)