Research collaboration guidelines

When I start a new research collaboration, I ask that we agree on a set of guidelines for practical details. I use this document as a starting point, and stand by these as the basis for new projects.

Push-button evaluation

All experiments should be runnable all at once with a *single command*. Example: `python3 eval.py`. This command may take arguments to run a subset, run with different parameters, ect. The eval published in a paper is the one with no arguments. It should generate all graphs used in the paper and all numbers (as latex macros) used as well.

The rationale for this requirement is that:

- Good science requires reproducibility.
- All members of the team should easily get eval results.
- An artifact for artifact evaluation is needed at some point anyways.

Communication in the project channel

In-person work is the best, but it's not always realistic or convenient. Projects should usually use a channel (Slack/ect) heavily, with a healthy project having active communication from every member regularly.

The channel is used for updates, questions, discussion, in-progress ideas, etc. More messages in the channel is almost always better, and the messages can be casual.

A typical usage pattern to aim for is at least one message a day, with many more on some days.

Code review is priority 1

Reviewing code is more important than other tasks. All changes are reviewed by another member of the team. Sometimes we try to move fast and incur tech debt when writing research code, but a review can at least make it clear what is being sacrificed.

Eval code doing any measurements should be carefully reviewed, as bugs in this code are very bad.

Minimize bash

Bash or shell code should be avoided, even at significant effort. Bash is difficult to refactor, extend, debug, test, and document.

No environment variables

Do not introduce environment variables. They cause installation issues and bugs. Sometimes it's absolutely necessary to set them for external tools, in which case we set them explicitly in the code of the project.

Relational data format for evaluation

Store experimental results and intermediate datasets as **tables** (lists of rows) without nested data. Usually, **one table** is enough for a project.

This has a few advantages

- It's easy to add new axes (columns) later.
- It's easy to fuse datasets

- It causes fewer bugs due to changing the format, unclear data format, or complex query code

Example JSON:

```
[
  {
    "benchmark": "benchA",
    "treatment": "optX",
    "seed": 42,
    "compile_time_seconds": 10.1231231,
    "commit": "a1b2c3d",
    "machine": "lab01",
  },
  {
    "benchmark": "benchA",
    "treatment": "optY",
    "seed": 42,
    "compile_time_seconds": 2.4324331,
    "commit": "a1b2c3d",
    "machine": "lab01",
  }
]
```

Avoid premature optimization

Make it work, optimize later when needed. Often it’s best to keep both a simple version of an algorithm and an optimized one, measuring the difference. If the simple one is far too slow, run it on a small set of benchmarks.
