Prediction Module Pitch

Model Management

Vasken Dermardiros

March 14, 2022

Problem

> The raw idea, a use case, or something we've seen that motivates us to work on this

This time around, the problem is to put the meat on the solution rather than building something from the ground up.

Background

Unlike ControlKit 1.0, ControlKit 2.0 is based on FastAPI and running the "main loop" is actually a single call.

Figure 1: FastAPI Endpoints

How does it work? Quite simply, a call is made to the ControlKit endpoint which runs a few sequential processes. It starts by getting and compiling the data, passes it to the ControlModule (algos), get back a sequence of controls, maybe passes that to the PredictionModule, gets predictions, passes it to the SafetyModule which will confirm if all is good and then pushes this to the proposed controls database (we may not write all the commands all the time). The process reads specific settings from the ConfigModule along the way as required.

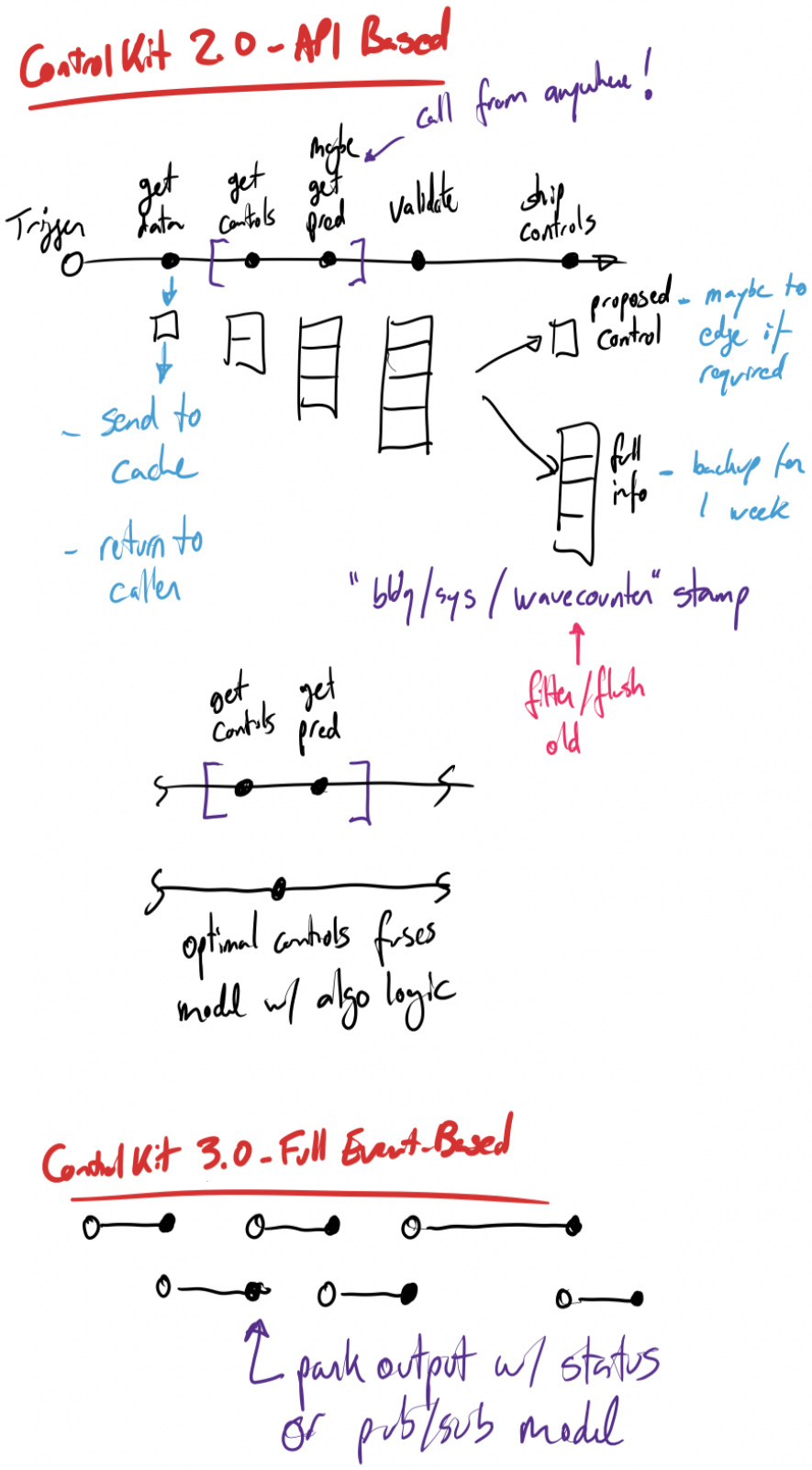

Figure 2: ControlKit 2.0 and 3.0

The difference between ControlKit 2.0 and 3.0 is that 3.0 is event-based where each phase or action is completely independant of the former. Each time an action is complete, the action would broadcast this state where some subscribed module(s) may pick it up from there. This is much trickier to orchestrate and we don't have the infrastructure now to do things like this. Would require a much tighter integration with DataStreams and full Kubernetes (k8s) deployment.

Lastly, so what's the trigger? It is a script that checks the `extraction_cycles` table for new data and once satisfied makes the call. Down the line, we may have the edge device itself make this call when it sees that its `tape` is either running out or the predictions are diverging. The `tape` is the two-or-more-hours of predictions and controls that will be sent to the edge device which will then write to the building one after the other. Alternately, the `tape` may be a decision tree that is updated regularly.

This API and triggered approach solves the data synchonizing issue and allows to have different versions or setups of ControlKit running for a given building in parallel. It allows to run a staging version of ControlKit in "virtual mode" which could be swapped into "control mode" for a short while. This simplifies our rollout strategy. We may also have a certain configuration used for special cases such as demand response or have specific algos running, and so on. This would reduce the need to develop a multi-control module.

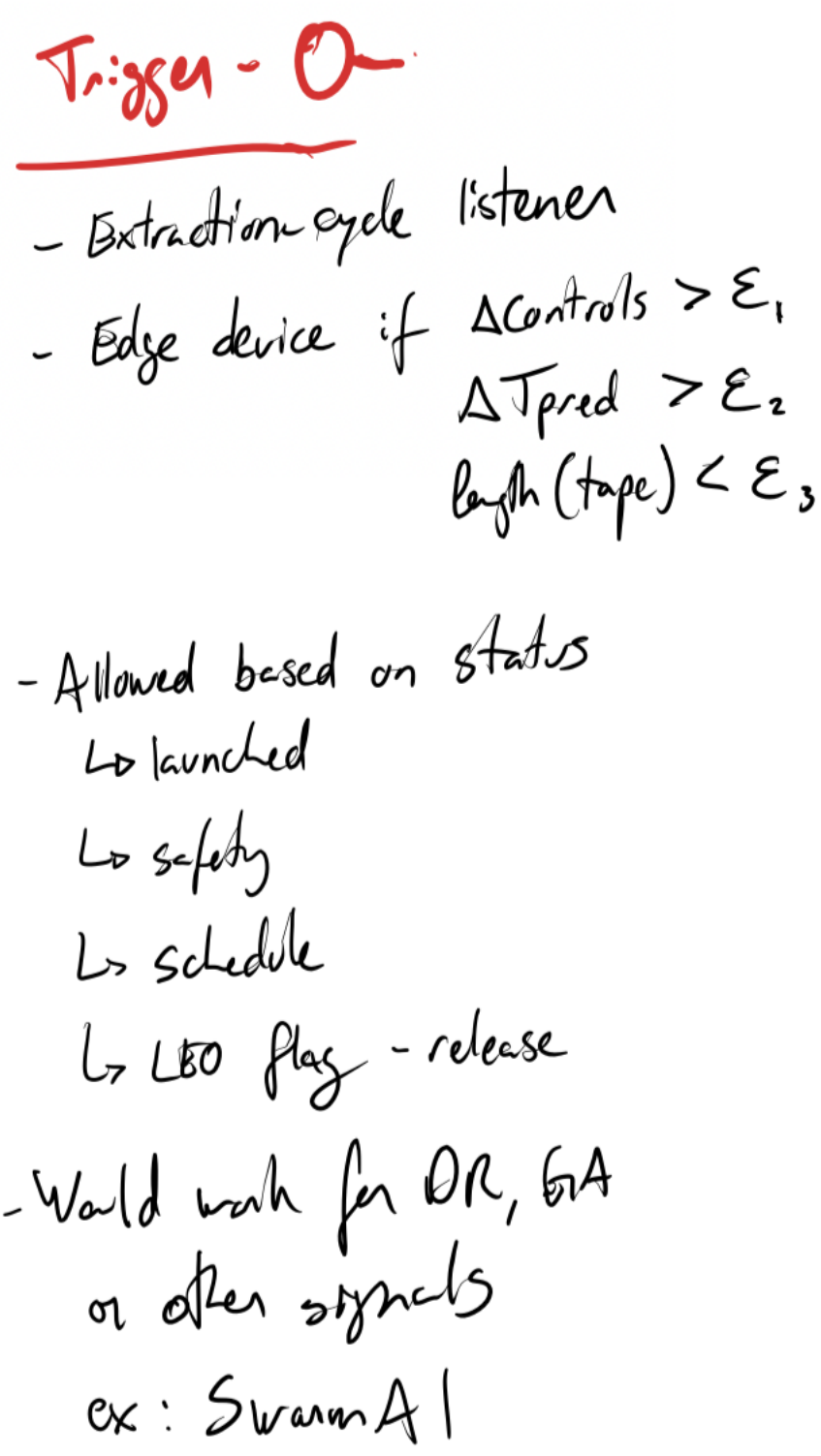

Figure 3: Trigger to call ControlKit endpoint

Main Problem

Which now brings us to the main problem in this pitch: an algo walks into a bar and wants a drink. The cocktail is a prediction. And unlike how we were doing things before, the algo is the first-class citizen here. The algo decides what model it needs (in terms of XUWO points) and the model will reflect the algo. Basically, if a zone goes missing, both the algo and model will be released for that zone or floor **but nothing more than that.** We shouldn't release a whole building if the janitor's closet's temperature sensor in the second basement is broken.

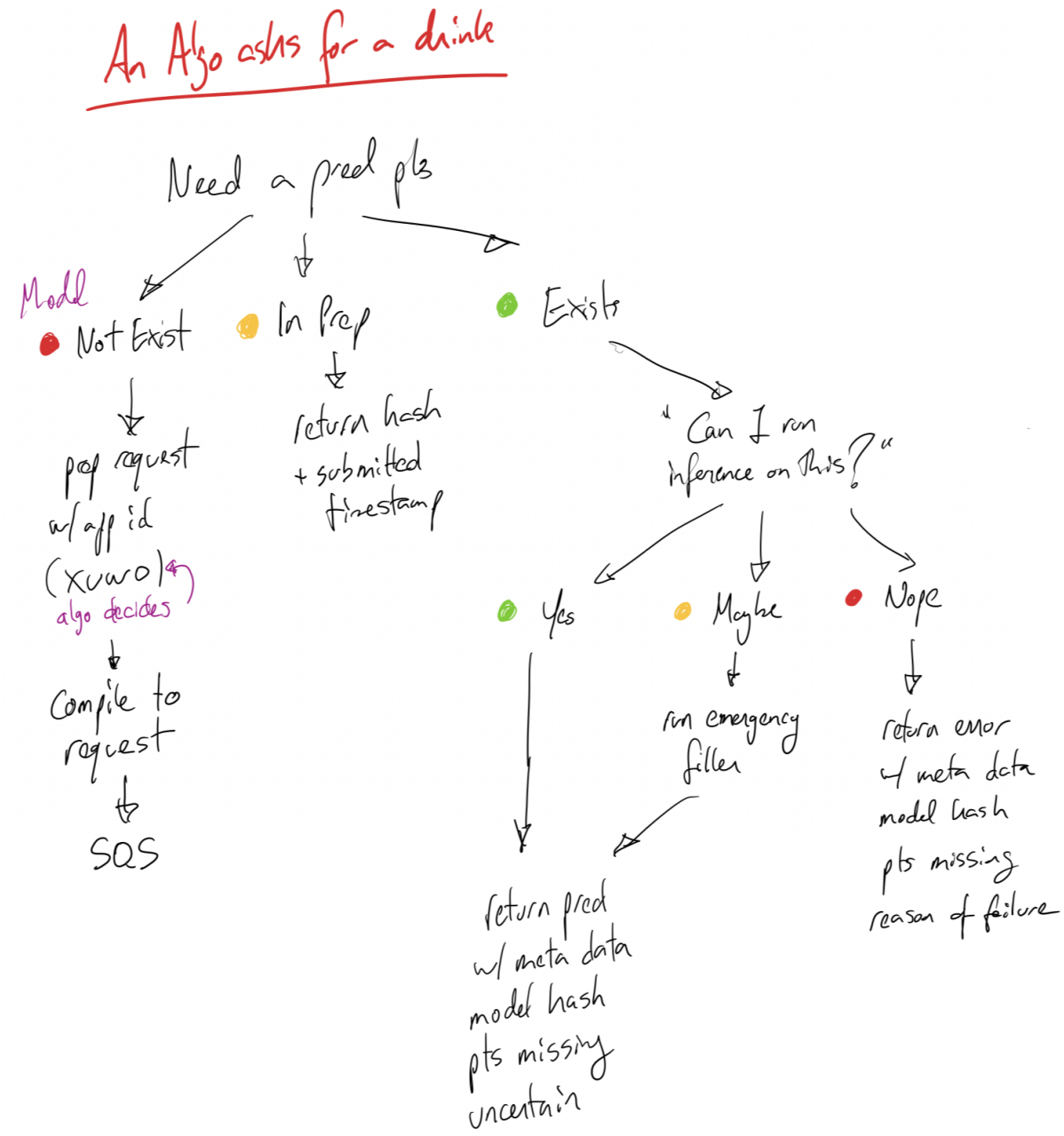

Figure 4: An algo walks into a bar

To obtain a prediction, a few logic steps have to be followed. On one side is how to make the calls and bring the data to the model. On the other is how to manage the "bar" to have the models ready. And these are the two pitches for this build cycle.

This pitch covers the latter.

Clarification on Submodels

Since the dawn of time, we've been building a model per building. The whole building. The `ai_config` does include the building hierarchy such that we could break the large model into smaller submodels based on levels, data quality, etc. using the `phd_submodel_generator()` function inside `DataKit.parsers`.

The days of taking the initiative are over. Ok. At least on this topic.

For ControlKit 2.0, the design is around having a parallel instance per each system / application running. This system will start by being the whole building, but as the algos are recoded, the algos will be deployed on a system-level and not a building-level. Being deployed on a system-level, the models that will serve these algos should follow the same structure and contain the same points in them. So, we will have a model per system / application. The extraction list may also follow this logic too, so that everything is broken up based on system / application. Of course, we will have multiple of these per building.

An algo configured for a system / application will know the points associated to itself and can pass it on in its model creation request.

Appetite

> How much time we want to spend and how that constrains the solution

Time budget: six weeks starting March 14, 2022.

Solution will allow getting predictions from the models through ClearML directly. The predictions will need to take less than two minutes to be processed completely to allow time for fetching data and pushing it to the controls table.

No need to be concerned about how the raw data is fetched, but will be provided a means to get it or get to it; and no need to be concerned about how the predictions are consumed.

On the model management side, please use as much as possible all the tooling in ClearML to publish and archive models. There are many services already and a lot of the heavy lifting was done in the last Build cycle.

For serving models, Emilio had explored this part in detail in the past. I hope you remember things or wrote things down in a remarkable way.

Solution

> The core elements we came up with, presented in a form that's easy for people to immediately understand

The solution contains seven parts: (1) defining what is a request, (2) able to generate a model given a request – regardless of the amount of data, (3) maintaining this request over time and updating as the data changes, (4) serving this model, (5) providing access to these models, (6) maintaining the model collection, and, (7) allowing for different levels of experimentation to simplify the development of new techniques.

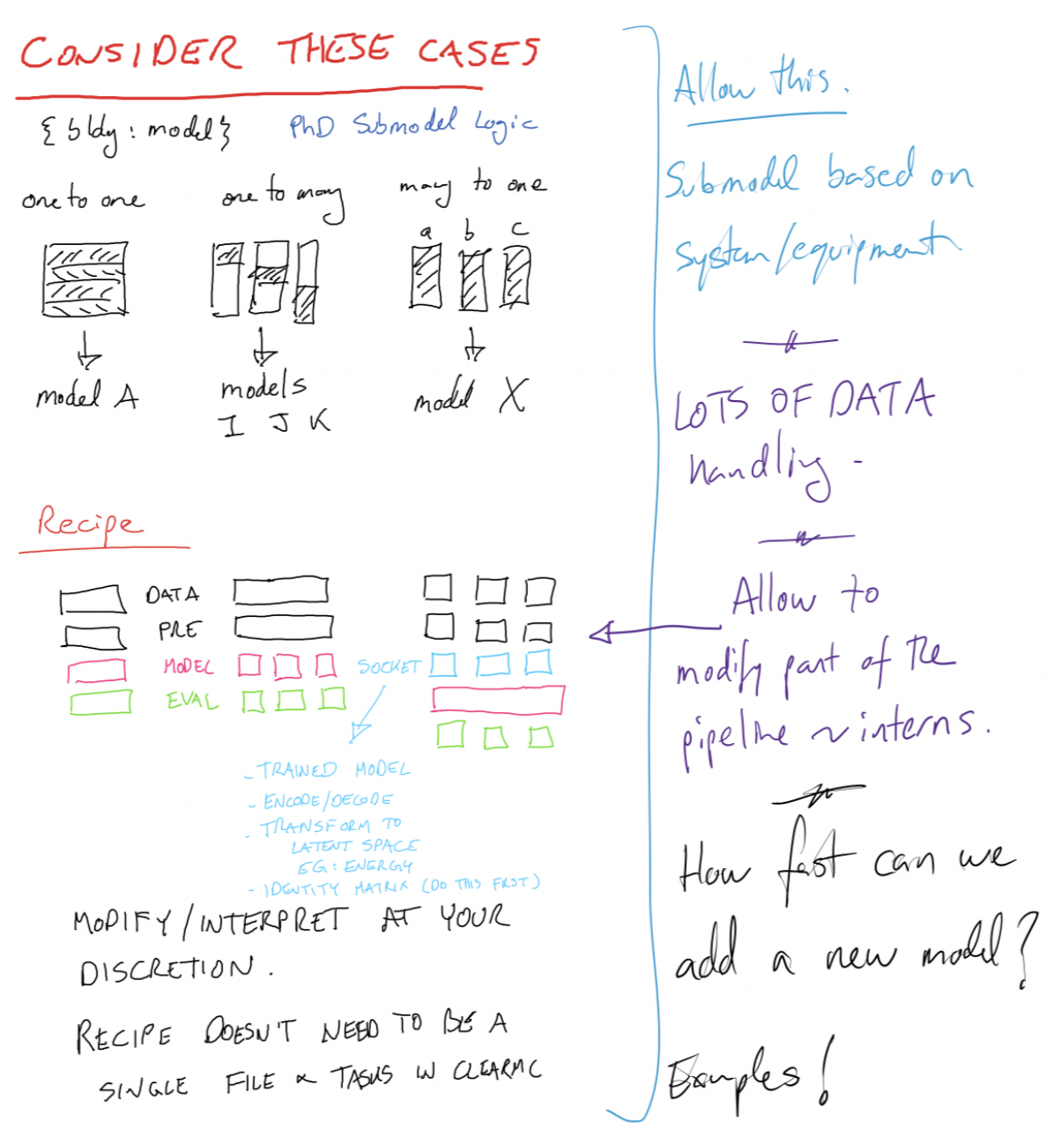

Figure 5: Add to submodels and allowing experimentation

Levels of Experimentation

The levels of experimentation depends on the user and how far it is compared to the standard pipeline.

- **Basic**: able to change date ranges and XUWO lists.
- **Advanced**: able to change pre-processing step to allow other input transformations, eg. dT vs T, or encode HVAC states.
- **Pro**: able to test a new model, needs to be in Python, but can be in Jax or PyTorch.

Metrics and reporting should work. The report generation would be another Task. User is responsible to provide code to apply inverse transformations.

The need comes from ONB, interns and friends wanting to try different inputs and study their output. They're not sure which input to use and whether they make a difference or not, eg. two ai_config files for two models, which is better? Think of this as a user-guided model search. The more ONB knows about the model and data, the better they can communicate with the client.

Able to fetch a trained model weight and run locally. FIXME: API call would return s3bucket location.

Compare and Select

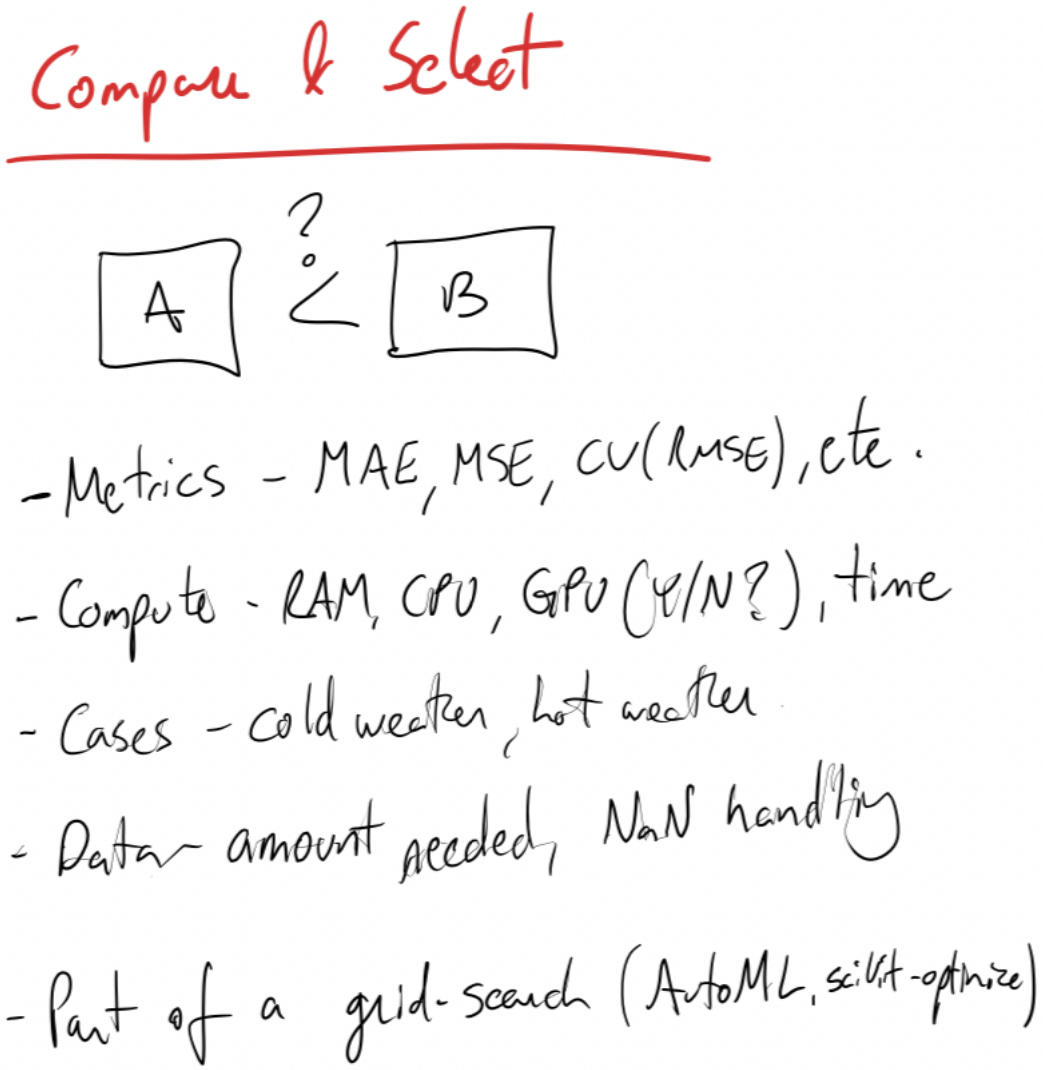

Figure 6: Compare and Select

Grid search can be seen as a *nice-to-have* because we wouldn't be running this for every building. We may be okay just running a long experiment (after the build cycle) to see what works and what doesn't. We can try to use the tools already in ClearML for AutoML, but my hunch is that we might not need to be that thorough every time.

MLOps Metrics

These are some MLOps metrics that we should be able to have numbers for and will be forwarded to the API. There could be more.

- Model update frequency met (and able to identify stale models)
- Time to train a model: request-to-serve, train-to-serve
- Time to re-train and deploy a new model and push to production
- Time to test and push new approach to production
- Performance of model endpoint for online predictions (e.g. response time in ms)
- Nmbr of calls / % of failed calls to the model endpoint
- Resource usage: CPU, GPU, memory, disk, etc.
- Queue time

Rabbit Holes

> Details about the solution worth calling out to avoid problems

For the request part, it should be as-is in the current deployment where the request is put into the SQS and consumed by ClearML. The Controller and Tasks spawned by this request will be different than the current deployment but the procedure should be comparable.

For building the report, provide a template Task and use only the functions already in PredictionModule. Allow future development by other people or interns.

NOTE Project is quite large as-is. Need to constrain it more.

No-Gos

> Anything specifically excluded from the concept: functionality or use cases we intentionally aren't covering to fit the appetite or make the problem tractable

Don't reinvent the ClearML API endpoints. See if there's something already in their documentation.

NOTE Project is quite large as-is. Need to constrain it more.

Notes that helped shape this pitch

> These are notes and not deliverables!

```
# Chat from call "Access to Hogwarts"
+ People: Ysael, David, Taso, Nicolas, Vasken
+ Date: Jan 11, 2022

[9:51 a.m.] Ysael Desage
So just to summarize what I understood in text, let me know if I missed
    something, but the big points are:

1. Show onboarding the new functionalities coming to Mission Control
   for the models (performance tracking, visualization etc.). Discuss
   from there if there is anything missing.
2. Give Nic and others who want to experiment code to train/test
   locally deep learning temperature models
3. Discuss longer-term desired structure for everyone, as new
   temperature prediction module is being reworked

Need: lack of transparency with what the models are predicting, how
   they work, what they look like and what kind of inputs affect the
   output.

ONB would also like to be able to *play* with the model to be able to
   better explain to the client what is going on.

# Need
1. Able to fetch a trained model weight and run locally: DevOps team
   only
2. API to have broader access; potentially external clients
3. Don't break anything going forward. Ever!


[12:17 p.m.] Anastasios Papachristou
Also, if we had the tool that we discussed with Nicolas the other day,
   we could easily do a test
2 models with 2 different config files




- [x] model hosting: https://developer.nvidia.com/nvidia-triton-
    inference-server
- [ ] have we had buildings that haven't gotten through because the
    models suck?
- [ ] can we send 20 request and see what happens?
- [ ] CK integration test: send a candidate building, trained for a
    certain period

- [ ] review performance of models
  - [ ] add Os in history
  - [ ] the different OPMs

```

```
37
38 # Vasken
39 + add rename_columns function in request -> where
40 + data flow example in satellite. which part of the data is created/
     handled where?
41 + monitor performance -> see everything in one view
42 + handle triggers
43 + handle schedules
44
45 + Seeing multiple of a same model...
46
47
48
49
50 + Collaboration between the multi-disciplinary team (e.g. data
     scientists, engineers, IT-Ops, legal, business)
51 + Attendance of key stakeholders to regular project updates (for
     example, every six weeks)
52 + Infrastructure scales to the machine learning teams without manual
     work from Ecosystem/DevOps/IT
53
54
55 + Scheduler
56   + I wrote some code and can be found here: https://git.brainboxai.net
       /Toolkit/ControlKit/src/branch/feature/schedule_maintenance/
       ControlKit/schedule_utils.py
```