StagedML: ML model library and management system

StagedML briefly

StagedML is an OpenSource project at it's early stage of development. It's author is Sergey Mironov, the former Huawei contractor (Sergey Mironov mwx579795, CBG Open Source Insight team)

- GitHub page https://github.com/stagedml/stagedml

The goal of this project is to extend a Functional Programmer's approach to the design of **Software Deployment System,** to the domain of Machine Learning applications.

The project was stated at Jan, 2020.

Software Deployment problem review: Traditional approach

Software deployment in general - is all of the activities that make a software system available for use [1]. Typically it addressess steps like building, packaging, distributing, installing and updating the software.

There is a long history of software deployment solutions. The list of well-known solutions for traditional software include:

- Linux from scratch http://www.linuxfromscratch.org
- Debian APT https://en.wikipedia.org/wiki/APT_(software)
- Gentoo Portage https://wiki.gentoo.org/wiki/Portage
- Windows installer https://en.wikipedia.org/wiki/Windows_Installer
- .. and so on

Traditional systems more or less solve the problem for their domain at the time of their creation. With the advances in the IT industry, the disadvantages of such systems became clear. The most important problems are:

- **Variants of «Dependency Hell» problem [2].**
- **Lack of security, need of ROOT access for the operator**
- **Unrealiability of updates, updates are non-crash-recoverable.**
- **Poor support of source-based distribution**

[1] - https://en.wikipedia.org/wiki/Software_deployment
[2] - https://en.wikipedia.org/wiki/Dependency_hell

Software Deployment problem review: Modern Approach, NixOS family

The next generation of Software Deployment systems were developed to combat the limitations of traditional systems. The new list includes:

- NixOS https://nixos.org (Proposed in 2003, In production since 2008)
- GNU Guix https://guix.gnu.org (In production since 2012)
- Docker https://www.docker.com
- AppImage https://appimage.org
- Etc.

All those systems address traditional deficiencies by using different set of technologies. In this presentation we focus on the family of NixOS systems marked by dashed frame. They are characterized by:

- Usage of immutable data managers for storing the software packages
- Usage of Turing-complete domain-specific languages for whole-system configuration
- Low requirements for the host system (run on pure POSIX core while Docker requires virtualization)
- Atomic full-system updates
- Continuous build farms (NixOS Hydra https://nixos.org/hydra/)

Software Deployment problem review: Machine Learning specifics

Recent advances in Deep Learning introduce new challenges and call for new solutions. Software now includes Machine Learning components which significantly differ from traditional software in many ways. Among others, they tend to re-define terms of ‘Building’, ‘Linking with Libraries’. ML components now have their specific kind of ‘Compilers’ and hybrid computing cores to execute.

Some of today’s systems that feature Machine Learning software deployment are:

- TensorFlow Hub (https://www.tensorflow.org/hub/)
- HuggingFace Models (https://huggingface.co/models)
- DeepPavlov (https://deeppavlov.ai)

For the author’s opinion, while providing some functionality for ML, they tend to re-introduce some of the deficiencies of traditional systems. They don’t receive a wide adoption in the domain of ML libraries yet and in additional, they don’t seem to integrate well with existing non-ML-aware systems.

The next slide demonstrates the problem using a popular ML library by Google as an example:

Software Deployment problem review: Criticism of existing ML library design

Screenshot of the GitHub Readme page of official BERT, by Google

We demonstrate the limitations of existing ML library's design on the example of one of IT leaders – the Google's "**TensorFlow official models**" located at https://github.com/tensorflow/models/tree/master/official/nlp/bert

The library is distributed in a source form, without clear separation in components. It's usage require form users many manual actions, as mentioned on the library's documentation page.

Google:

This section describes one of several steps that you have to run manually

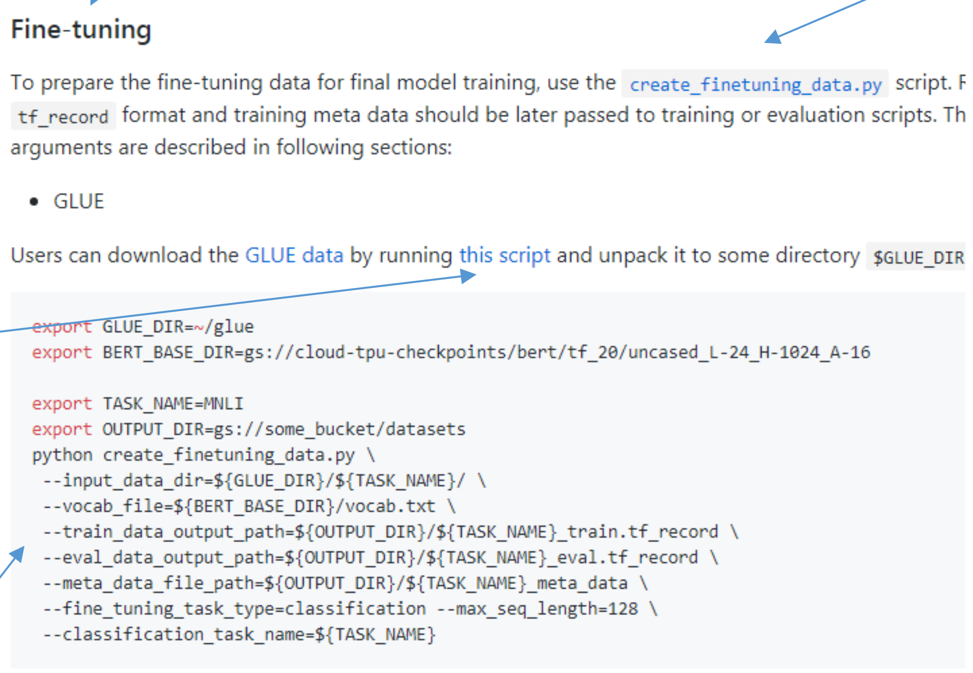
Download and run this script!

Note that other datasets require other actions.

But run this script first!

Fill those environment variables

Fill those command line arguments



Fine-tuning

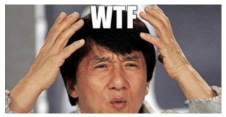
To prepare the fine-tuning data for final model training, use the `create_finetuning_data.py` script. F
`tf_record` format and training meta data should be later passed to training or evaluation scripts. Th
arguments are described in following sections:

- GLUE

Users can download the GLUE data by running this script and unpack it to some directory `$GLUE_DIR`

```
export GLUE_DIR=~/glue
export BERT_BASE_DIR=gs://cloud-tpu-checkpoints/bert/tf_20/uncased_L-24_H-1024_A-16

export TASK_NAME=MNLI
export OUTPUT_DIR=gs://some_bucket/datasets
python create_finetuning_data.py \
 --input_data_dir=${GLUE_DIR}/${TASK_NAME}/ \
 --vocab_file=${BERT_BASE_DIR}/vocab.txt \
 --train_data_output_path=${OUTPUT_DIR}/${TASK_NAME}_train.tf_record \
 --eval_data_output_path=${OUTPUT_DIR}/${TASK_NAME}_eval.tf_record \
 --meta_data_file_path=${OUTPUT_DIR}/${TASK_NAME}_meta_data \
 --fine_tuning_task_type=classification --max_seq_length=128 \
 --classification_task_name=${TASK_NAME}
```

Users:



Introducing StagedML

https://github.com/stagedml/stagedml

StagedML is a library of ML models which includes facilities for training and experimenting. Those facilities are focused on providing the level of quality comparable with a NixOS-family deployment system.

Features of StagedML which are similar to NixOS:

- Both are based on the immutable data management API
- Both provide turing-complete language for whole-pipeline configuration
- Low requirements for host system: core <2K lines of Python code, needs only a bare Python3 interpreter + standard libraries for running
- Atomic updates within the file system
- Backward-compatibility with NixOS (potentially, not done yet)

Features of StagedML which are not present in NixOS:

- StagedML doesn’t have such a large set of packages, that NixOS has
- StagedML is based on Python because ML apps do largely use this language
- Unlike NixOS, StagedML does support non-deterministic build (here, train) outcomes, which are important for Machine Learning.

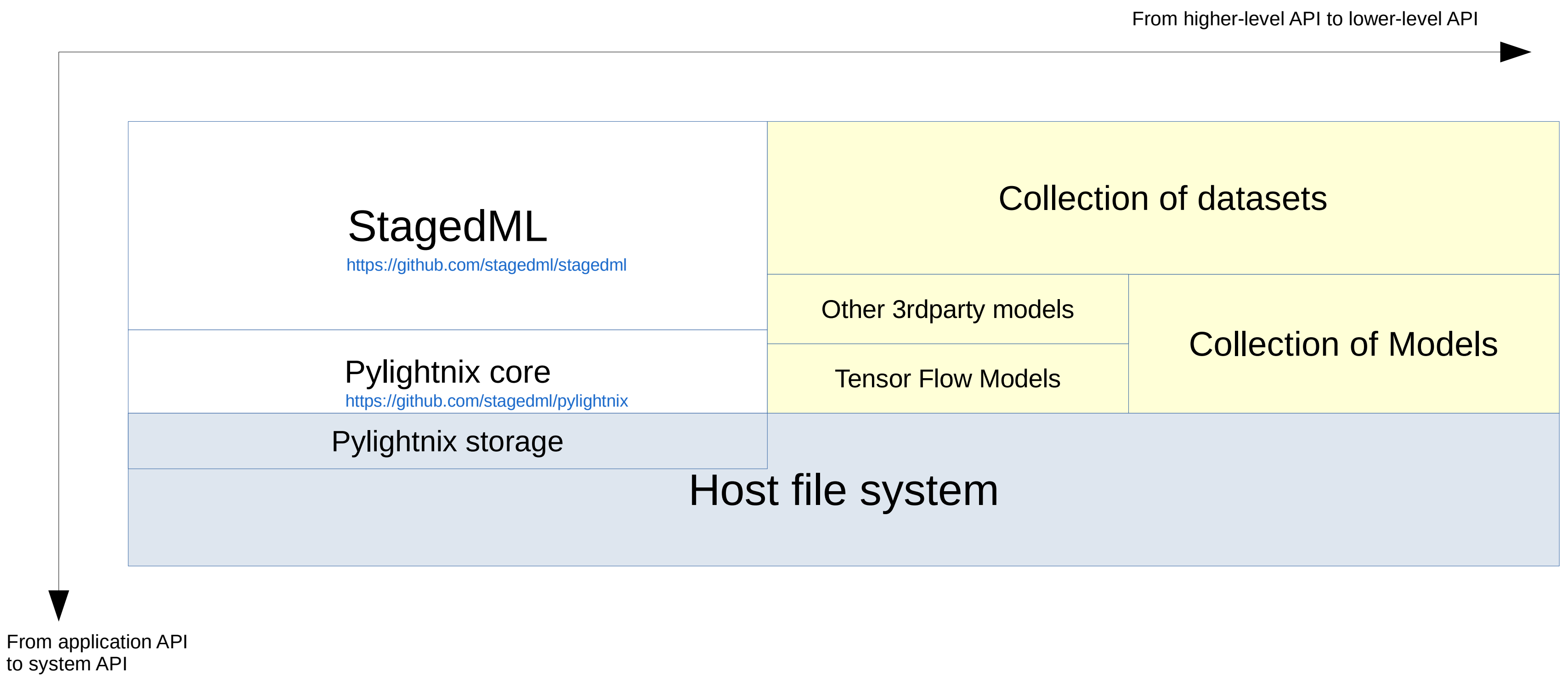
Missing features (lack of resources/not needed now):

- Lack of build isolation, and strong parallelism
- Lack of continuous build and deployment system (not done yet)

Introducing StagedML

Components overview:

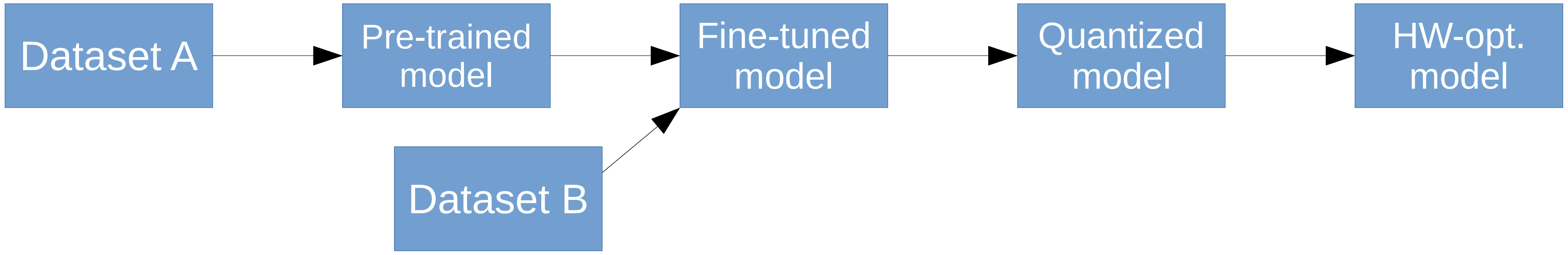
- **Pylightnix core** offers mid-level API which defines types of entities, formalizes two-pass building process, provides REPL tools, etc.
- **Collection of models and datasets** wraps third-party models and datasets into Pylightnix stages, which in turn become components of future pipelines
- **StagedML** glues stages together into a set of top-level pipelines


From higher-level API to lower-level API
StagedML
https://github.com/stagedml/stagedml
Collection of datasets
Other 3rdparty models
Collection of Models
Tensor Flow Models
Pylightnix core
https://github.com/stagedml/pylightnix
Pylightnix storage
Host file system
From application API to system API

How does StagedML work?

Programmer defines pipeline by defining Stages. Stages are entities consisting on Configuration (JSON dict), Matcher and Realizer algorithms as defined by the core. Programmer then registers Stages in the **dependency resolution manager** and obtains handles called **Derivation Reference**. He/she may then pass those references from stage to stage to mark dependency relations.

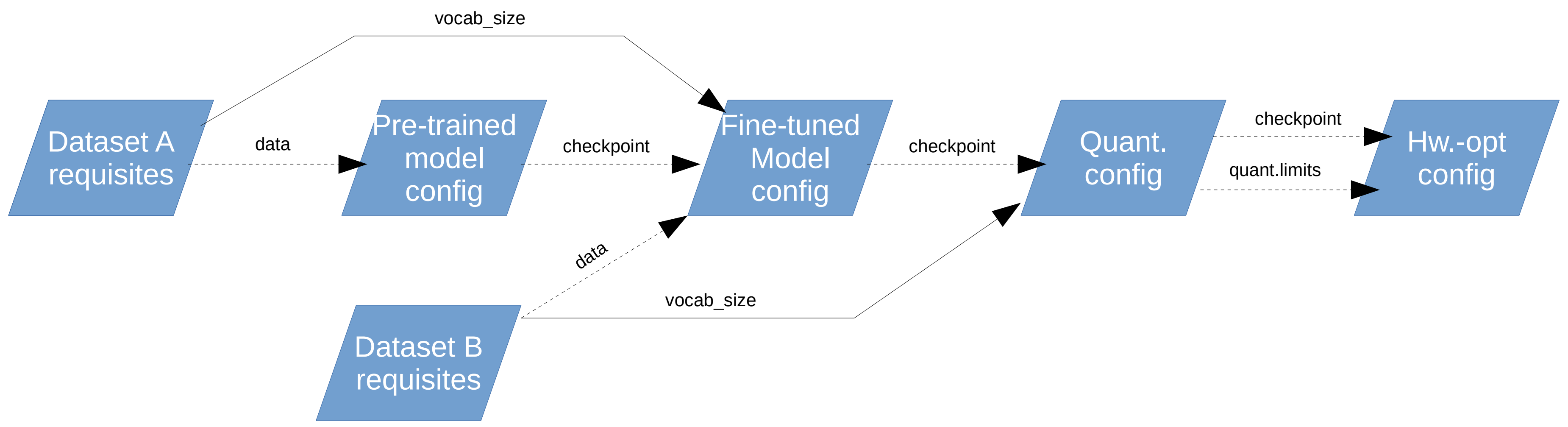
Consider a pipeline of BERT-based classification. Our goal is to compute weights of the final Hardware-optimized model.



Assuming that all components are properly defined, the StagedML would compute weights of the final model by using the procedure shown on next slides.

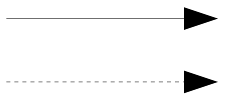
How does StagedML work?

1. StagedML runs first pass and computes the configuration of the whole pipeline.



Every stage may access upstream stage's configuration parameters, define it's own configurations and also declare future output values via **promise values**. At this pass users have an opportunity to catch many errors which involve misspelled parameter names and non-matching numeric values.

Legend:

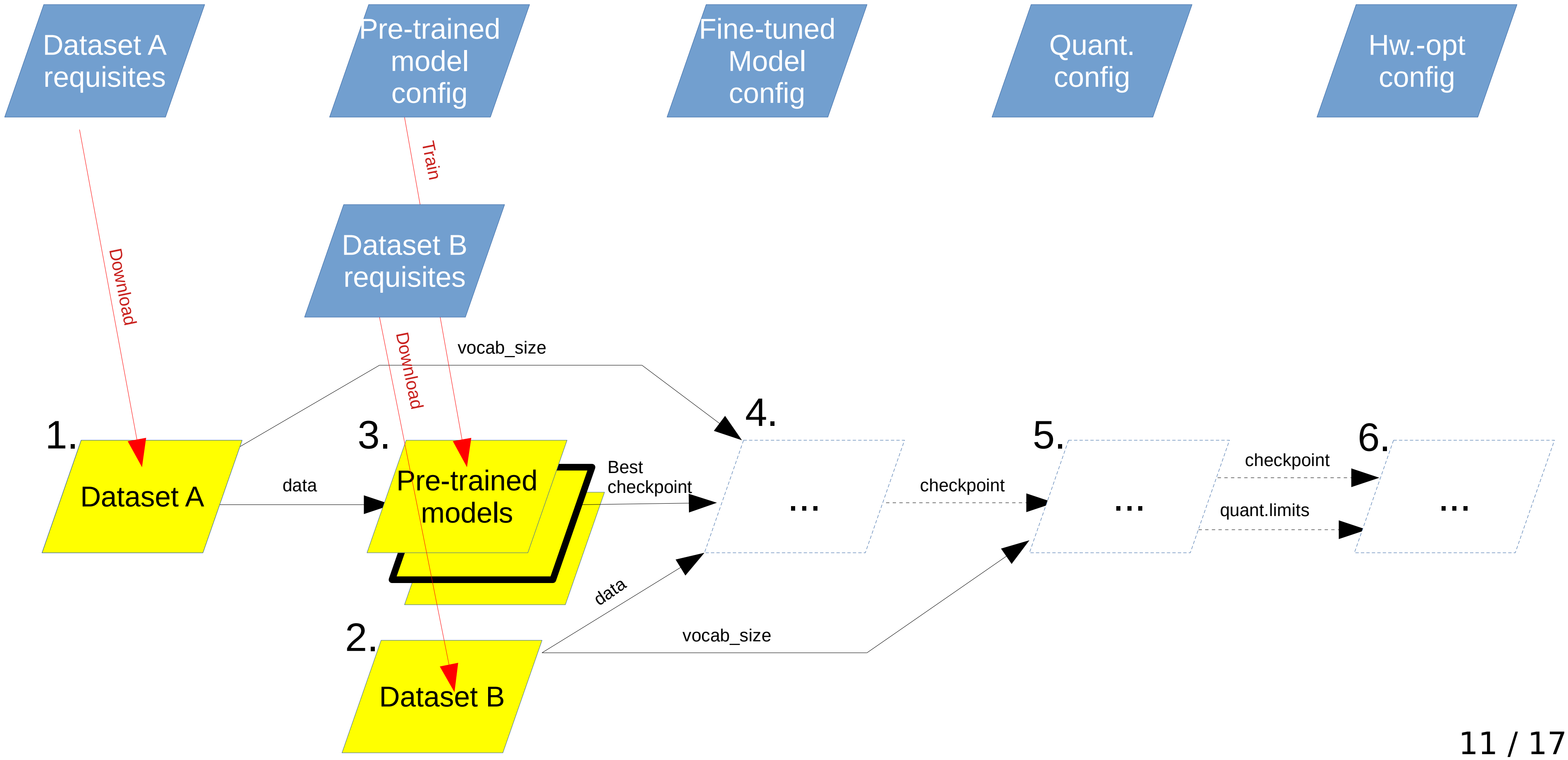


Usage of configuration parameter

Usage of promise value

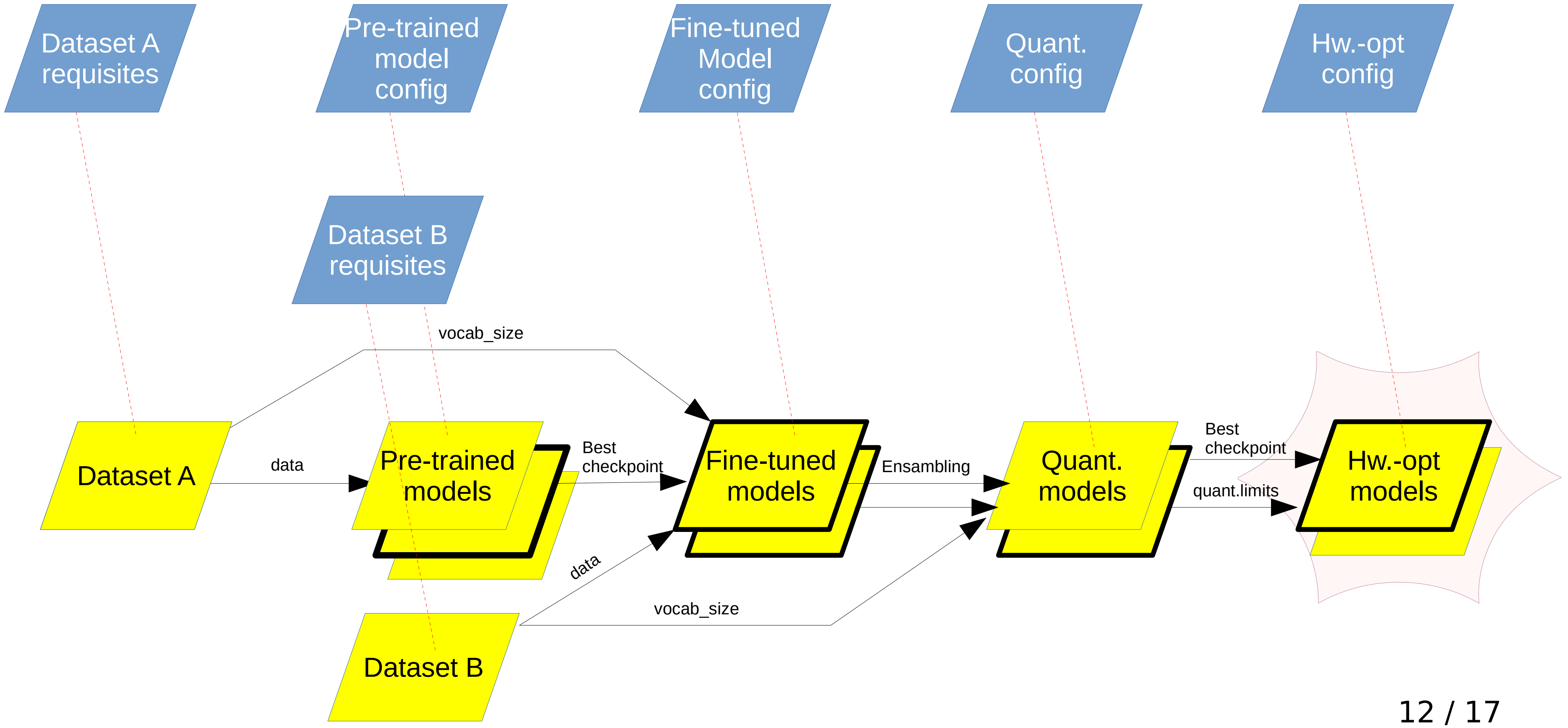
How does StagedML work?

2. Then StagedML runs second pass by asking stages to **realize** themselves in right order. For every realization the framework checks that it's promises are fulfilled. StagedML does support multiple realizations for one stage, which makes non-trivial ensambling scenarios possible.



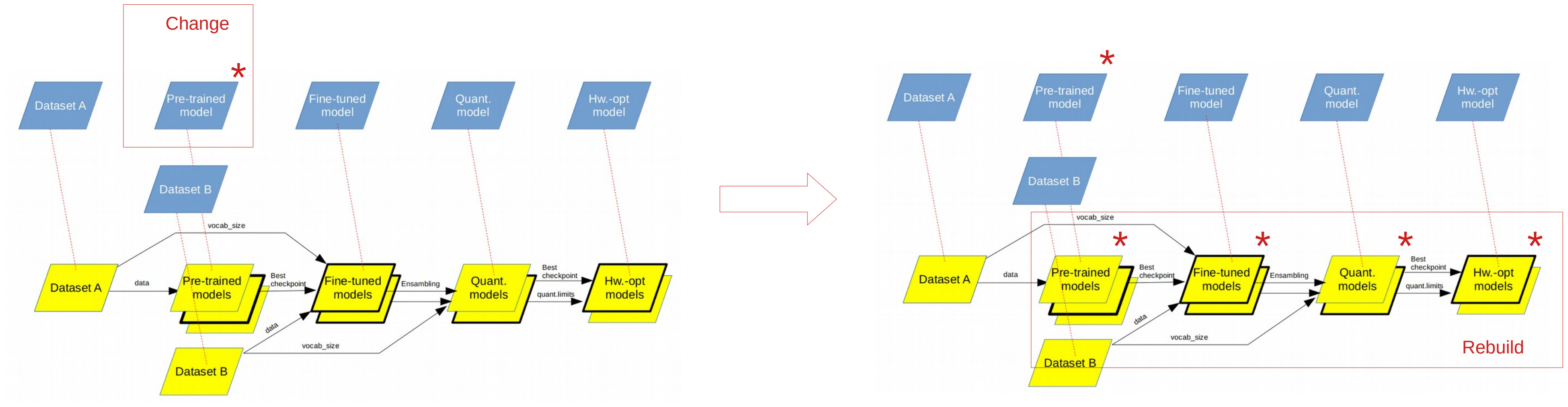
How does StagedML work?

3. Finally, when the pipeline is fully realized, we may use data from any stage of the graph. At this point the model is marked as ready to be deployed.



How does StagedML work?

4. Immutability of data means that no change in configuration may stay unnoticed. For any change, StagedML knows which stages to rebuild.

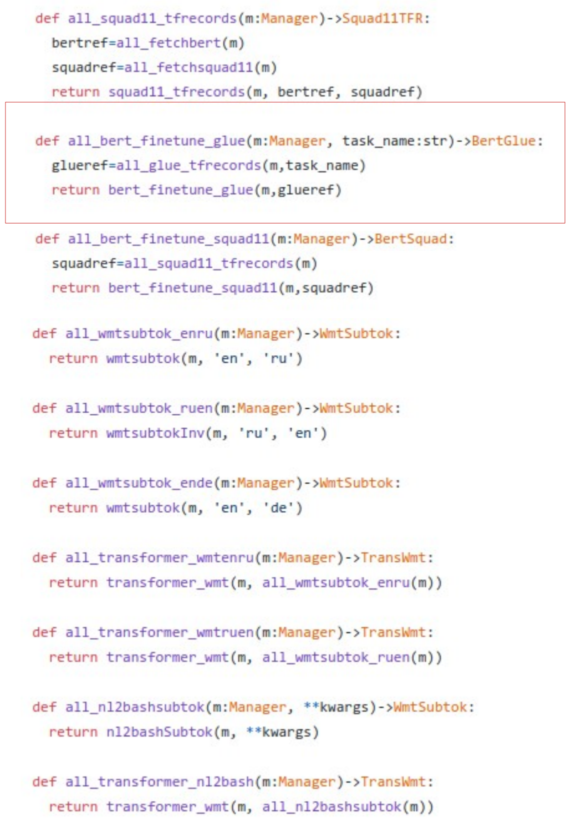


5. After rebuilding, the storage contains both new and old versions of the data. In order to delete old versions, the **garbage collector** should be run explicitly. Running garbage collector is the only top-level action which may remove data.

Proposed ML library design

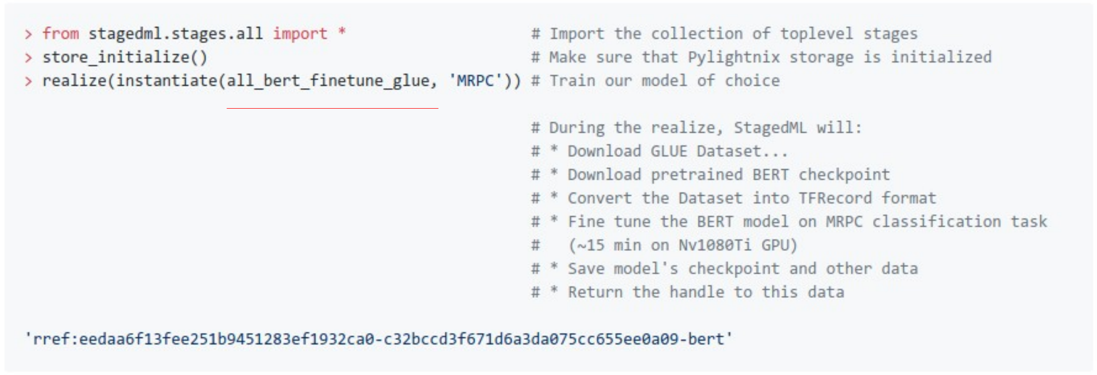
With StagedML, obtaining the desired model is easy

1. Select the pipeline config



```python
def all_squad11_tfrecords(m:Manager)->Squad11TFR:
  bertref=all_fetchbert(m)
  squadref=all_fetchsquad11(m)
  return squad11_tfrecords(m, bertref, squadref)

def all_bert_finetune_glue(m:Manager, task_name:str)->BertGlue:
  glueref=all_glue_tfrecords(m,task_name)
  return bert_finetune_glue(m,glueref)

def all_bert_finetune_squad11(m:Manager)->BertSquad:
  squadref=all_squad11_tfrecords(m)
  return bert_finetune_squad11(m,squadref)

def all_wmtsubtok_enru(m:Manager)->WmtSubtok:
  return wmtsubtok(m, 'en', 'ru')

def all_wmtsubtok_ruen(m:Manager)->WmtSubtok:
  return wmtsubtokInv(m, 'ru', 'en')

def all_wmtsubtok_ende(m:Manager)->WmtSubtok:
  return wmtsubtok(m, 'en', 'de')

def all_transformer_wmtenru(m:Manager)->TransWmt:
  return transformer_wmt(m, all_wmtsubtok_enru(m))

def all_transformer_wmtruen(m:Manager)->TransWmt:
  return transformer_wmt(m, all_wmtsubtok_ruen(m))

def all_nl2bashsubtok(m:Manager, **kwargs)->WmtSubtok:
  return nl2bashSubtok(m, **kwargs)

def all_transformer_nl2bash(m:Manager)->TransWmt:
  return transformer_wmt(m, all_nl2bashsubtok(m))
```

2. Ask StagedML to realize the config



```
> from stagedml.stages.all import *                     # Import the collection of toplevel stages
> store_initialize()                                    # Make sure that Pylightnix storage is initialized
> realize(instantiate(all_bert_finetune_glue, 'MRPC')) # Train our model of choice

                                                        # During the realize, StagedML will:
                                                        # * Download GLUE Dataset...
                                                        # * Download pretrained BERT checkpoint
                                                        # * Convert the Dataset into TFRecord format
                                                        # * Fine tune the BERT model on MRPC classification task
                                                        #   (~15 min on Nv1080Ti GPU)
                                                        # * Save model's checkpoint and other data
                                                        # * Return the handle to this data

'rref:eedaa6f13fee251b9451283ef1932ca0-c32bccd3f671d6a3da075cc655ee0a09-bert'
```

3. Use trained model(s)

- ✔ Review the configuration of every stage of the pipeline
- ✔ Turn stages into dependencies for other stages
- ✔ Publish models in the continuous integration system
- ✔ Deploy models to the user devices
- ✔ Compare with other realizations, prepare scientific reports

Who could benefit from the proposed technologies?

1) Scientists who work with Machine Learning, Genetic Algorithms or other processes which may produce non-deterministic outcome:
   - Pylightnix core API formalizes non-deterministic outcomes
     - Ensambling of machine learning models
     - Hypothesis-testing applications
   - Immutable data workflow makes sure that no data will be overwritten (at the cost of disk space which is cheap nowadays)
   - Pylightnix core allows us to inspect the configuration of a whole pipeline. It simplifies processing of experiment results and writing reports. StgaedML may be easily integrated with report generators like **Pweave** and **Jupyter notebook.** See refs. [4] and [5] on the next slide.
2) Application designers (?)
   - Pylightnix core may be used as a domain-specific package manager. For example, it may be used to ship new version of ML models to the client machines.
   - Deployment part needs to be added
3) Operators of cloud computing services (?)
   - StagedML could potentially be used as a backbone of continuous integration system which supports Machine Learning components
   - Support for specialized hardware needs to be added

Thank you

References:

1) https://github.com/stagedml/stagedml StagedML main page
2) https://github.com/stagedml/pylightnix Pylightnix core main page
3) https://github.com/stagedml/stagedml/blob/master/src/stagedml/stages/all.py
   StagedML model library
4) https://github.com/grwlf/ultimatum-game/blob/master/docs/Pylightnix.md
   Example of evolutionary algorithm implementation in Pylightnix
5) https://github.com/stagedml/stagedml/blob/master/run/nl2bash/out/Report.md
   Machine learning experiment in StagedML
6) https://discourse.nixos.org/t/pylightnix-a-lightweight-nix-like-dsl-in-python-for-ml/5629?u=sergeymironov
   NixOS forum announcement

Contacts

Sergey Mironov

E-mail: grrwlf@gmail.com
LinkedIn: https://www.linkedin.com/in/mironovsergey/