# AI in a Web-based Survey Instrument: A Low Latency, Real-time Prediction Serving Service

Andrea Roberson (andrea.roberson@census.gov)



## Background

- Deep Learning (DL) is being deployed in a growing number of applications which demand text categorization at the time of data collection.
- We discuss applying a DL model to the Annual Wholesale Trade Survey (AWTS) to code open-ended remarks.
- Continuous integration (CI) is a software engineering practice that helps a team manage the development life cycle. An agile development process is used to build, test, integrate, and deploy a DL application with CI.

## Goal

Present a novel system architecture for low-latency and real-time inference at scale for National Statistics Offices (NSOs)

## Challenges

- BERT (Bidirectional Encoder Representations from Transformers) is a DL model developed by Google. The BERT-base model contains 110M parameters.
- While BERT's performance is impressive, it is comparatively slow in terms of both training and inference. How can we reduce the size of these models?

## Methodology

**Distilled Deep Learning classification**

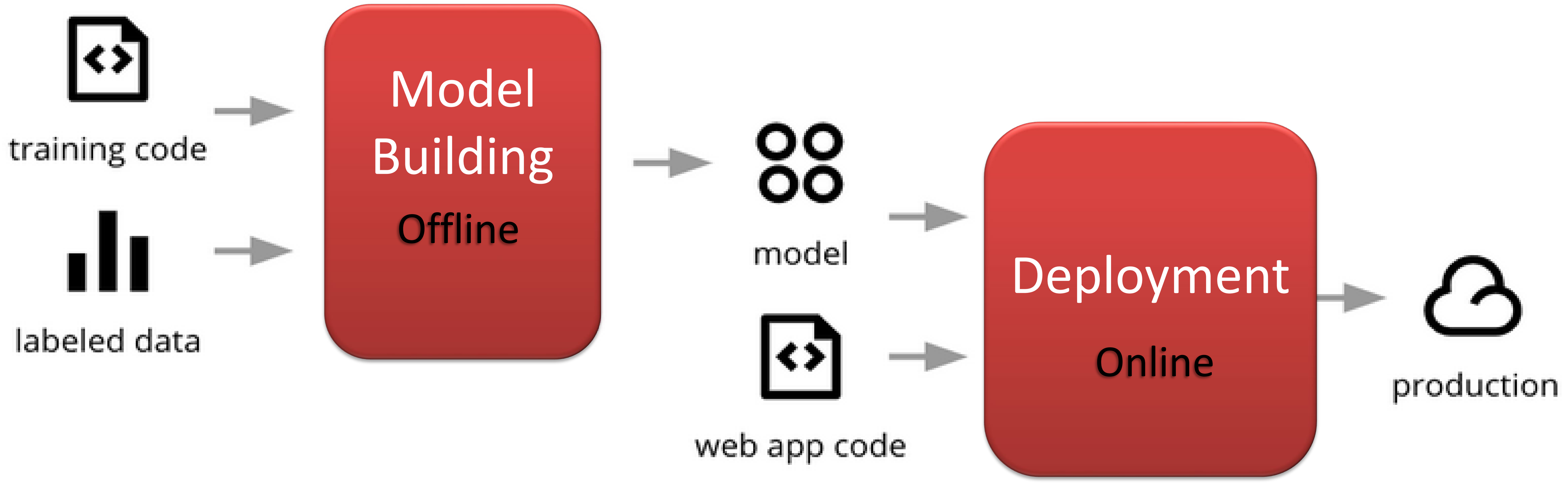
- We trained a DistilBERT model using the labeled AWTS remarks text, integrate that model into our web application, which is then deployed to a production server environment.
- Figure 1 shows our Python start function that keeps the deserialized model file in memory.

Figure 1

```
def start(name, predict):
#get the name of this api
    server_dir =
os.path.dirname(os.path.realpath(__file__))
    sasha_dir = os.path.dirname(server_dir)
    inbox = sasha_dir + "/jobs/" + name + "/inbox/"
    outbox = sasha_dir + "/jobs/" + name + "/outbox/"
    while True:
        found_job = look_for_jobs(predict, inbox, outbox)
        if not found_job:
            sleep(0.1) #Sleep for 100 milliseconds
```

- The python script looks in a folder we called inbox, and it looks for jobs. Before dropping into a while True loop, we have the model file in RAM. The prediction task begins when our function finds a job in the inbox and ends with the classified text being dropped into the outbox as a JSON file.

**Decoupled serving system to train our ML models, integrate them with a web application, and deploy into production**




## System Framework

- A primary goal of our serving system is to decouple applications from models
  - Allows DevOps team to focus on building reliable low latency applications.
  - Simplified the model deployment process for data scientists. Allows them to be oblivious to system performance and workload demands.

## Deployment Pipeline

**Our system requires a two-step DevOps process:**

- (1) ML developers commit code to a Git-versioned repository.
- (2) then a Jenkins Continuous Integration (CI) process builds, tests, and validates the most recent master branch. If everything meets deployment criteria, a Continuous Delivery (CD) pipeline releases the latest valid version of the model to customers.

# AI in a Web-based Survey Instrument: A Low Latency, Real-time Prediction Serving Service

Andrea Roberson (andrea.roberson@census.gov)

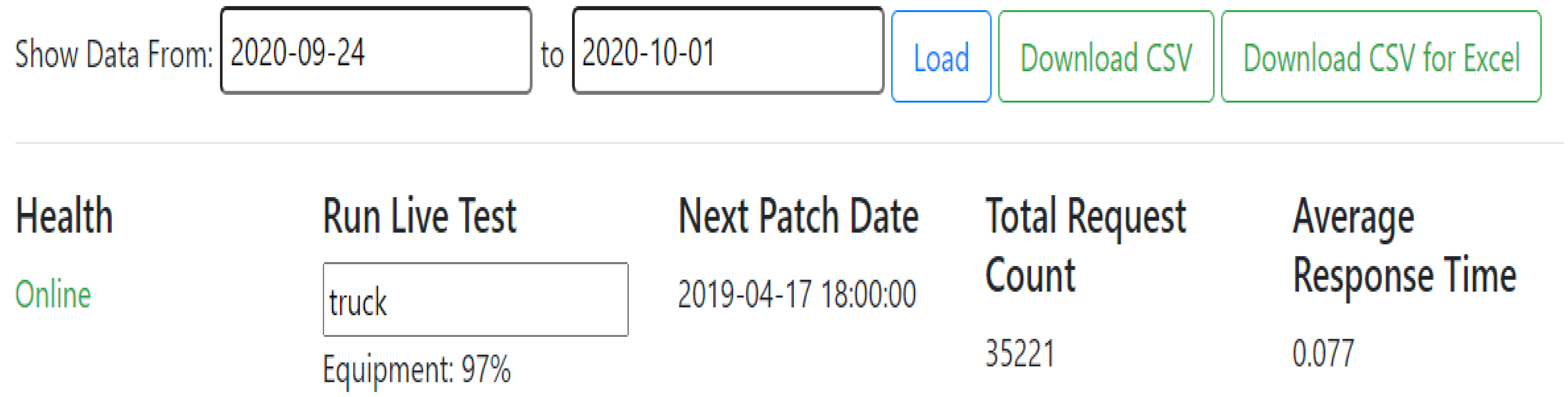


## Results

- After two years in production, results show the effectiveness of our proposed system design and deployment approach for classifying Annual Capital Expenditures Survey (ACES) open-ended text responses.
- The text categorization done at the time of data collection has significantly reduced the workload of staff, by 60% to 80% for manual review of written responses.

**The API reporting tool helps staff test the DL model's results**

To access data, you can use the UI below or the API




## Conclusions

- We have developed a CI system for integrating DL into production. We have validated our solution and operationalized it for the Annual Capital Expenditures Survey (ACES).
- Our hope is that this approach will significantly reduce the manual review of open-ended questionnaire responses.

## Current Work

- Next steps include:

  Quantization to improve the efficiency of DL computations through smaller representations of model weights.
  - Post-training: train the model using float32 weights and inputs, then quantize the weights.
  - Quantization-aware training: quantize the weights during training.

## References


- C. Renggli, B. Karlas, and B. Ding, Continuous Integration of Machine Learning Models with ease.ml/ci: Towards a Rigorous Yet Practical Treatment, SysML Conference (2019), 1–19.
- C. Sun, N. Azari, and C. Turakhia, Gallery: A Machine Learning Model Management System at Uber, EDBT Conference (2020), 474–485..

Andrea Roberson (andrea.roberson@census.gov)



# To access data, you can use the UI below or the API