Keynote
Goals - sustainable cloud computing - lower carbon emission in ML workload
Carbon aware solutions - compute in low carbon regions > have more renewable energy sources, i.e. solar, wind - compute in the time with most renewable energy generated - choose cloud provider with low PUE/ internal energy consumption of data center - choose efficient hardware, i.e. GPU, TPU - compute with special purposes, i.e. model pretraining/ model resizing
Energy intensive ≠ carbon intensive Depend on data center - using renewable energy > low/ no carbon computing
Current cloud computing has 2.5 - 3.7% GHG emission *GHG: greenhouse gas, e.g. CO_2, methane
Carbon emitting sources - coal - natural gas - oil
Non-carbon emitting sources - renewable energy, i.e. solar, wind
Carbon emission formula - Amount of carbon used in ML workload = Carbon intensity * Amount of Energy used in ML workload
Power usage effectiveness/ PUE - measure data center computing efficiency - **internal energy consumption of data center**/ real energy used
Low carbon region selection factors - cost - data center location > security - latency > performance
AI computing causes impacts - energy/ carbon: power data center - water: cool down for processors to run
Model computing ways - training - fine-tuning - inference/ serving
Inference - make predictions - generate text, images
Model types - generative model - transformer model - conversational model
Model/ task categories - text classification (single/ multi tasks) - token classification - image classification - text generation - image generation - object detection - image captioning - masked language modeling - extractive question answering (QA) (single/ multi tasks) - summarization (single/ multi tasks)
Model modality - text to category - text to image - text to text - image to text - image to category
Neural architecture: automate neural network design

Machine learning lifecycle - hardware, i.e. CPU, GPU, data center - training, i.e. pre-training, fine-tuning - inference, i.e. serve or interact with users in real time
Embodied carbon - any carbon emitted in supply chain for a given item - e.g. CPU, GPU, TPU, data centers, etc - GPU, TPU: more common and efficient for ML training - data centers need energy to power and cool down
Transfer learning - pre-training: from supervised model, with big datasets - fine-tuning: from pre-trained model, with smaller datasets
Prompt engineering: no need to train, fine-tune from scratch and get a model to use
Electricity map API - retrieve the last updated carbon intensity information in different regions by API call - alternative: Watttime
Electricity map API call process Task: retrieve data of carbon intensity and power breakdown of each energy source - import environment - load environment - select a training location, i.e. lat, lon - define Electricity map API_carbon intensity url with selected training location (Task 1) - import request - import helper - get request with carbon intensity url - import json - load json with defined request, i.e. API returned content - define Electricity map API_power breakdown url with selected training location (Task 2) - get request with power breakdown url - load json with defined request, i.e. API returned content *request function import from Python library *task 1 & 2 goal: return required data to compare energy consumption > low carbon or not *call power_stats in one step with a new training location

Power_stats one step code (after import and define settings)

```
import helper, requests, json, numpy as np
def power_stats(lat,lon, api_key=helper.load_emaps_api_key()):
    coordinates = { "lat": lat, "lon": lon }

    url_intensity = f"https://api.electricitymap.org/v3/carbon-intensity/latest?lat={coordinates['lat']}&lon={coordinates['lon']}"
    request_intensity = requests.get(url_intensity, headers={"auth-token": api_key})
    intensity = json.loads(request_intensity.content)

    url_breakdown = f"https://api.electricitymap.org/v3/power-breakdown/latest?lat={coordinates['lat']}&lon={coordinates['lon']}"
    request_breakdown = requests.get(url_breakdown, headers={"auth-token": api_key})
    breakdown = json.loads(request_breakdown.content)

    breakdown_abridged = {
        'renewablePercentage': breakdown['renewablePercentage'],
        'fossilFreePercentage': breakdown['fossilFreePercentage'],
        'powerConsumptionBreakdown': breakdown['powerConsumptionBreakdown'],
        'consumption_percent': {
            k: np.round((v/breakdown['powerConsumptionTotal']) * 100)
            for k, v
            in breakdown['powerConsumptionBreakdown'].items()
        },
    }

    return intensity, breakdown_abridged
```

API returned content of a selected training location - carbon intensity - datetime - updated at time - created at time - renewable energy amount/ type - power consumption amount *if need power consumption %, use numpy to count and print out

Training process in low carbon regions Criteria: need to use GCP Vertex Ai to choose low carbon regions - set up GCP, i.e. account info, project info/ ID - initialize Vertex Ai, i.e. task.py has all functions for training - define training steps in one code as a task.py --- import libraries --- create/ load a dataset --- create model --- train model - define a training location with low carbon intensity to train model, i.e. select by Electricity map API/ GCP carbon access doc - create storage bucket to store artifacts duing training process, i.e. store different project data in same selected training location - define custom training job, i.e. add back defined training location, task.py, storage bucket, job name - train model - delete bucket *Electricity map API: real time data *GCP: batch data/ historical data *if use Electricity map, need to call API
Training process in low carbon regions (use real time carbon intensity to select a training location) Basic process - get real time carbon intensity data > low carbon training location > low carbon training Detail process - set up GCP, i.e. account info, project info/ ID - initialize Vertex Ai, i.e. task.py has all functions for training - import environment - load environment - import request - import helper - import json - select the training location with real time lowest carbon intensity between Opt 1 and 2 - Opt 1: define carbon intensity function with selected training location, i.e. define url, get request, load json with API returned content - Opt 2: define cleanest function with region list, i.e. return the training location with real time lowest carbon intensity - define task.py, i.e. training project - define storage bucket - define custom training job, i.e. add back defined training location, task.py, storage bucket, job name - train model in selected training location - delete bucket to release resources *Opt 1: have a selected training location *Opt 2: select a training location based on real time lowest carbon intensity data *both Opt 1 and 2 use Electricity map *Electricity map real time data is for different cloud providers ; GCP batch data, i.e. Carbon Free Energy Percentage (CFE%) is only for GCP users *real time data providers: Electricity map, Watttime
task.py content - import libraries - create/ load a dataset - create model - train model
Google cloud footprint report - for GCP projects only - check the footprint by projects, year created in google cloud - can retrieve data by SQL in footprint dataset per account