Large Vision Models at Scale

A 3.5 week journey to serve hundreds of models in production

onfido

Agenda

onfido

1. **Intro**
   Who are we?
2. **Identity Verification**
   Why automate it?
3. **High Performance at Scale**
   How do we get there?
4. **Actionable LoRA**
   How can you use it?

Agenda

1 **Intro**
Who are we?

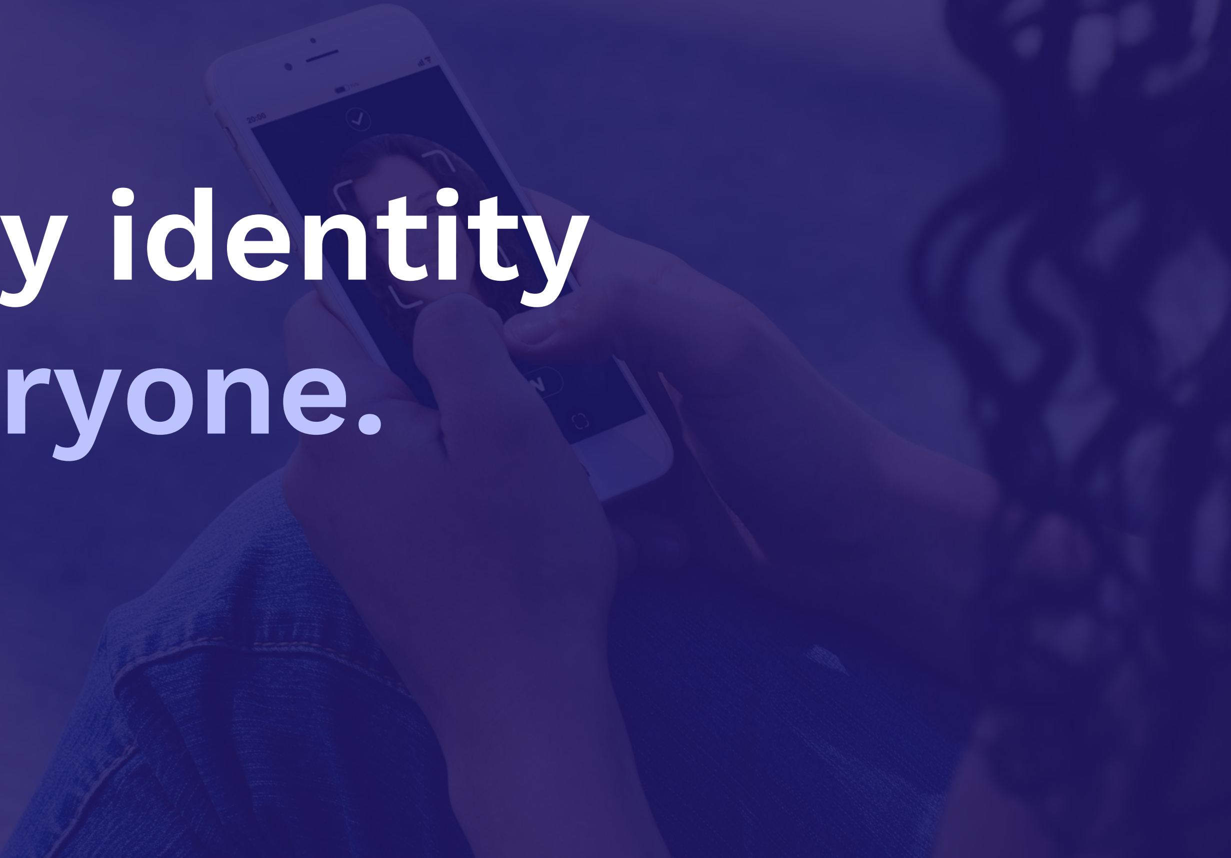
2 **Identity Verification**
Why automate it?

3 **High Performance at Scale**
How do we get there?

4 **Actionable LoRA**
How can you use it?

onfido

simplify identity
for everyone.



Onfido's 3 layers of identity verification

Do you have a **genuine** ID?

Are you a **real life** human?

Does your face **match** your ID?

1

2

3

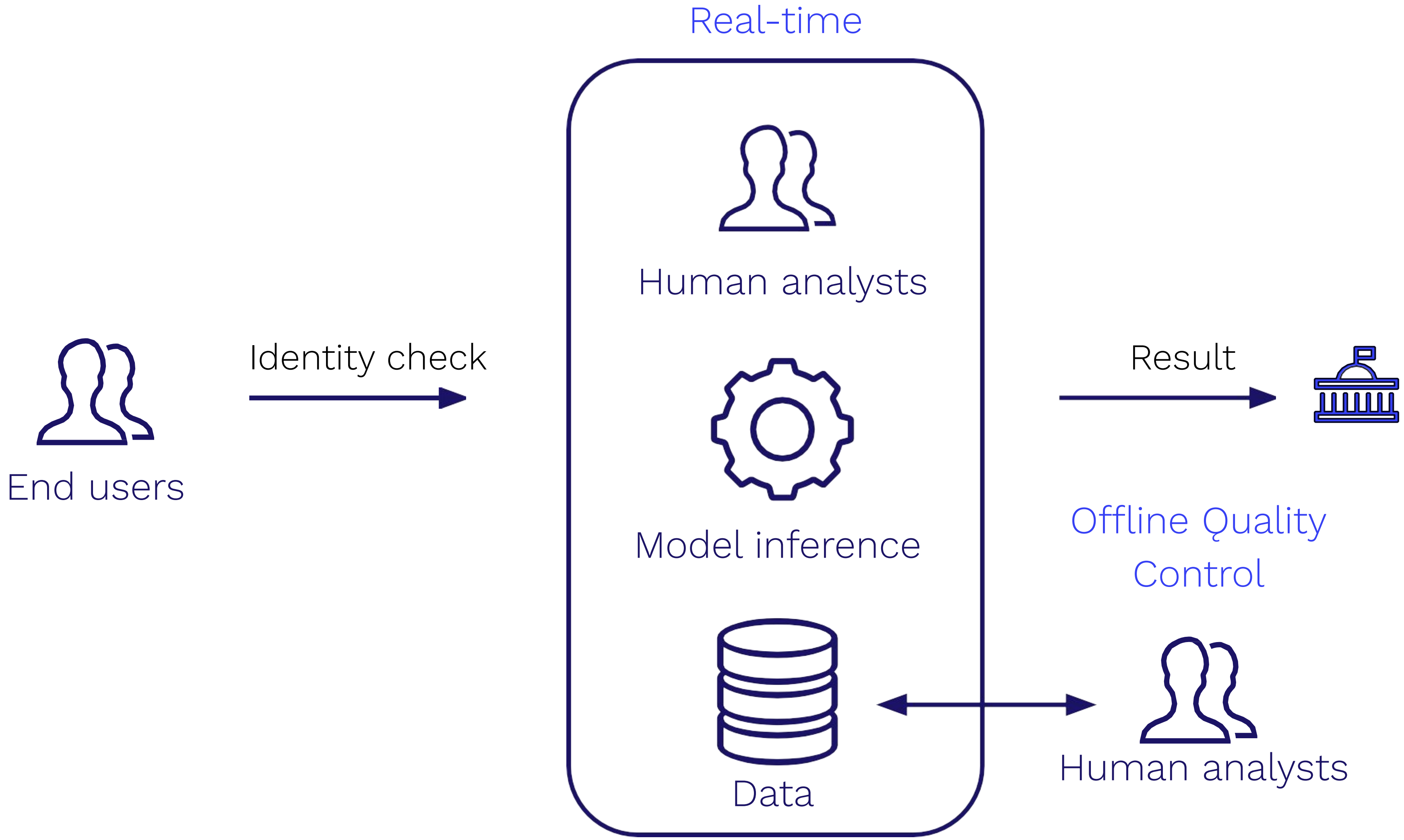
Agenda

1 **Intro**
Who are we?

2 **Identity Verification**
How is our system structured?

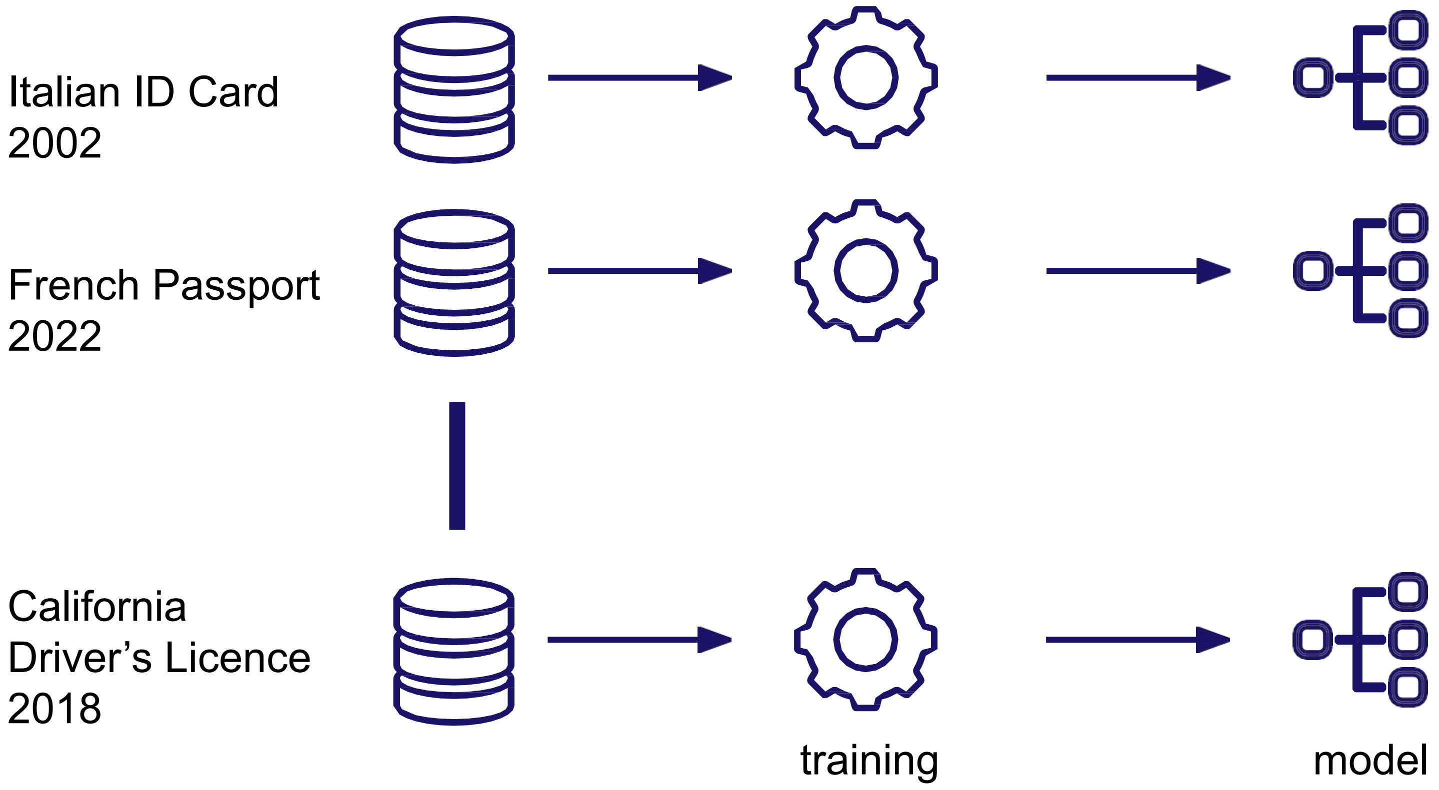
3 **High Performance at Scale**
How do we get there?

4 **Actionable LoRA**
How can you use it?


Real-time
Human analysts
Model inference
Data
End users
Identity check
Result
Offline Quality Control
Human analysts

Focusing on Document Verification

Current Status



Agenda

onfido

1 **Intro**
Who are we?

2 **Identity Verification**
Why automate it?

3 **High Performance at Scale**
How do we get there?

4 **Actionable LoRA**
How can you use it?

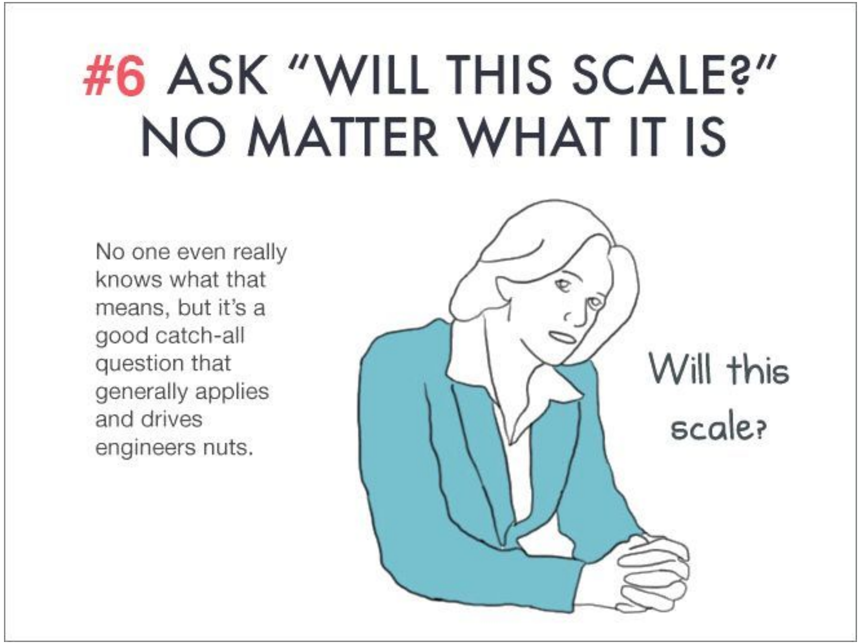
onfido

2x in accuracy

1. Transformer-based vision model
2. Single model per doc-type

Naive approach

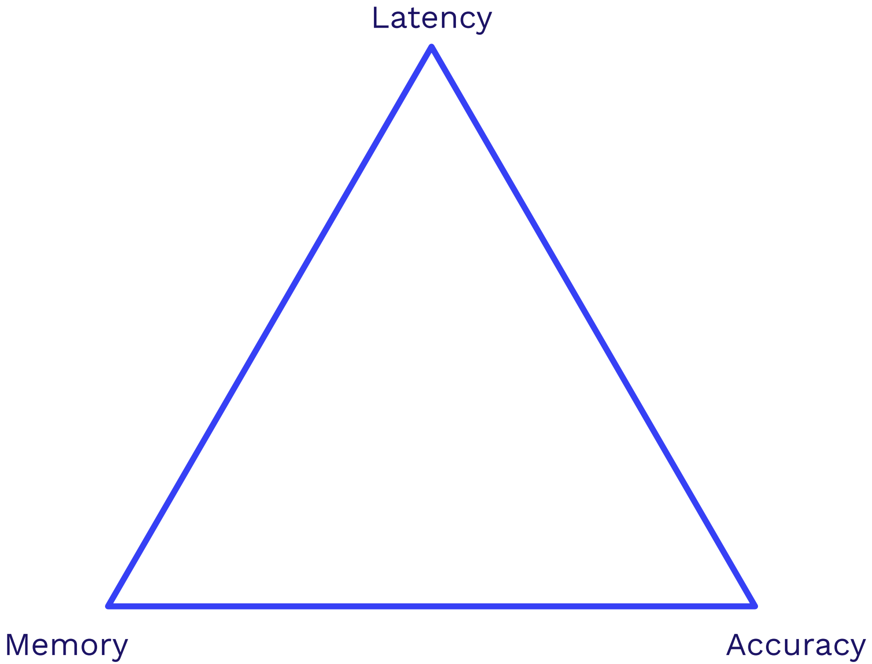
- Load the entire finetuned model onto the GPU at half precision
- ~4 models per GPU



From "Tricks to Appear Smart in Meetings"

bigco employee
onfido
@yacinemtb
h100

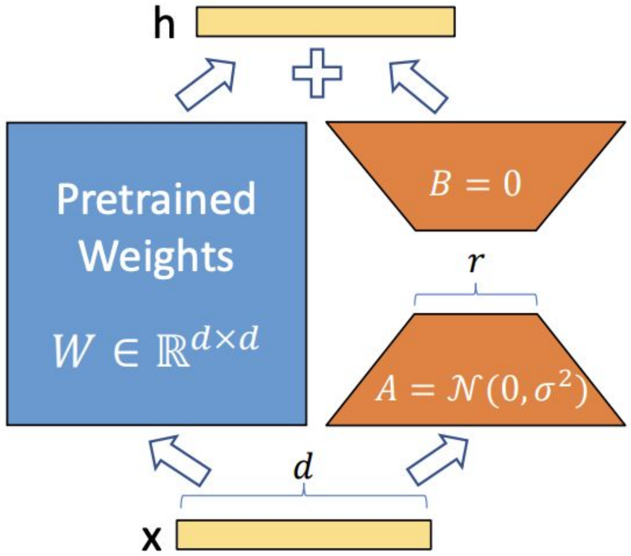
The Classic Trilemma



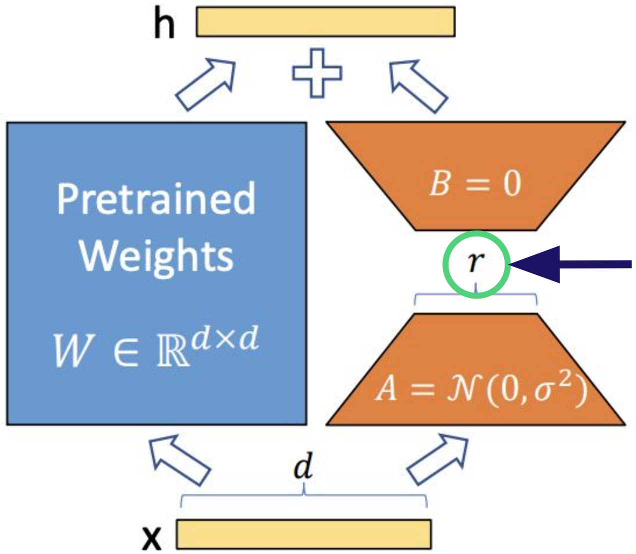
Memory

Enter LoRA

a PEFT technique



Enter LoRA


h
Pretrained
Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

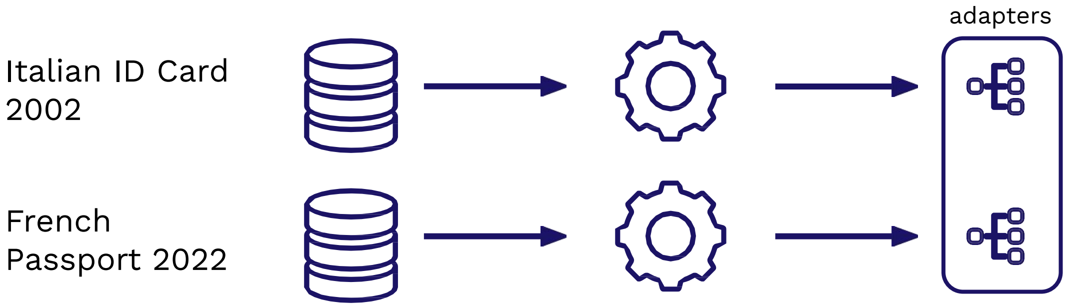
LoRA example

OS HF model

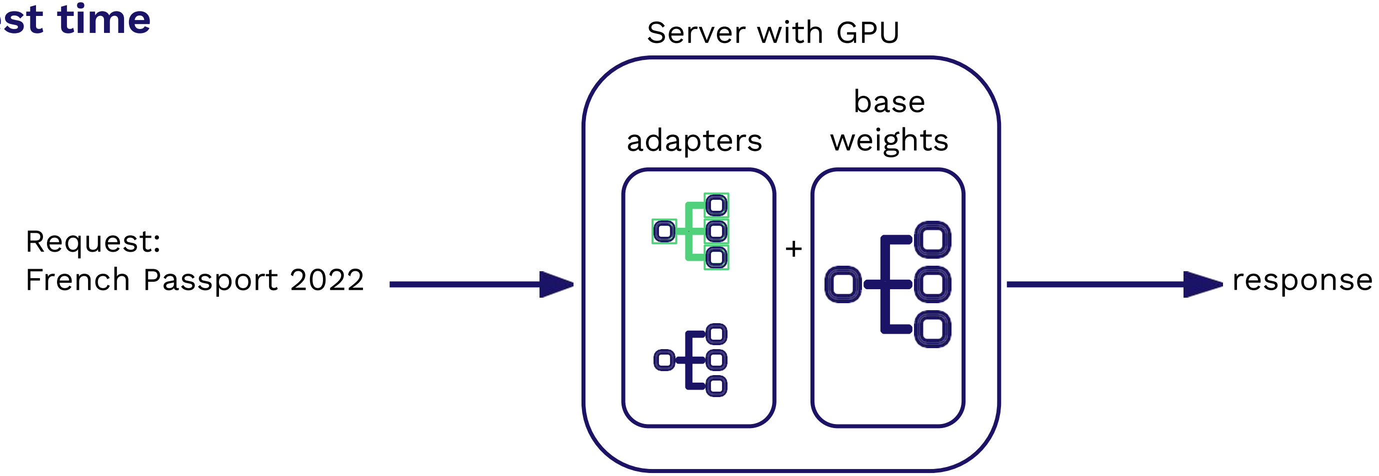
```
(encoder): ViTEncoder(
  (layer): ModuleList(
    (0-11): 12 x ViTLayer(
      (attention): ViTAttention(
        (attention): ViTSelfAttention(
          (query): Linear(
            in_features=768, out_features=768, bias=True
            (lora_dropout): ModuleDict(
              (default): Dropout(p=0.1, inplace=False)
            )
            (lora_A): ModuleDict(
              (default): Linear(in_features=768, out_features=16, bias=False)
            )
            (lora_B): ModuleDict(
              (default): Linear(in_features=16, out_features=768, bias=False)
            )
            (lora_embedding_A): ParameterDict()
            (lora_embedding_B): ParameterDict()
          )
```

https://github.com/lucapericlp/lora-latencies/blob/master/LoRA_adapter_merging.ipynb

Training time



Request time

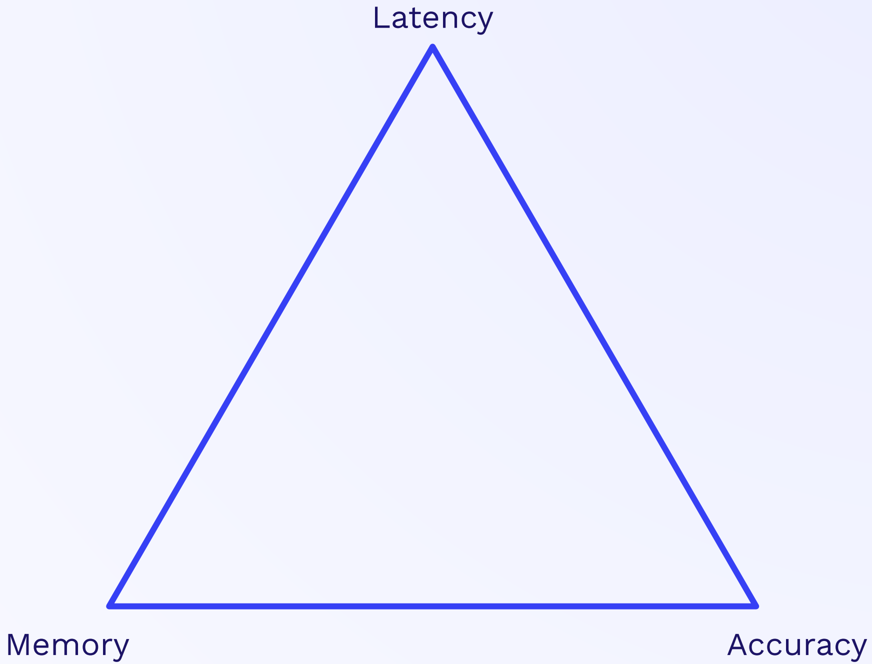


Solving the scaling problem

1. GPU node efficiency
2. Maximise flexibility

onfido

But what do the tradeoffs look like?

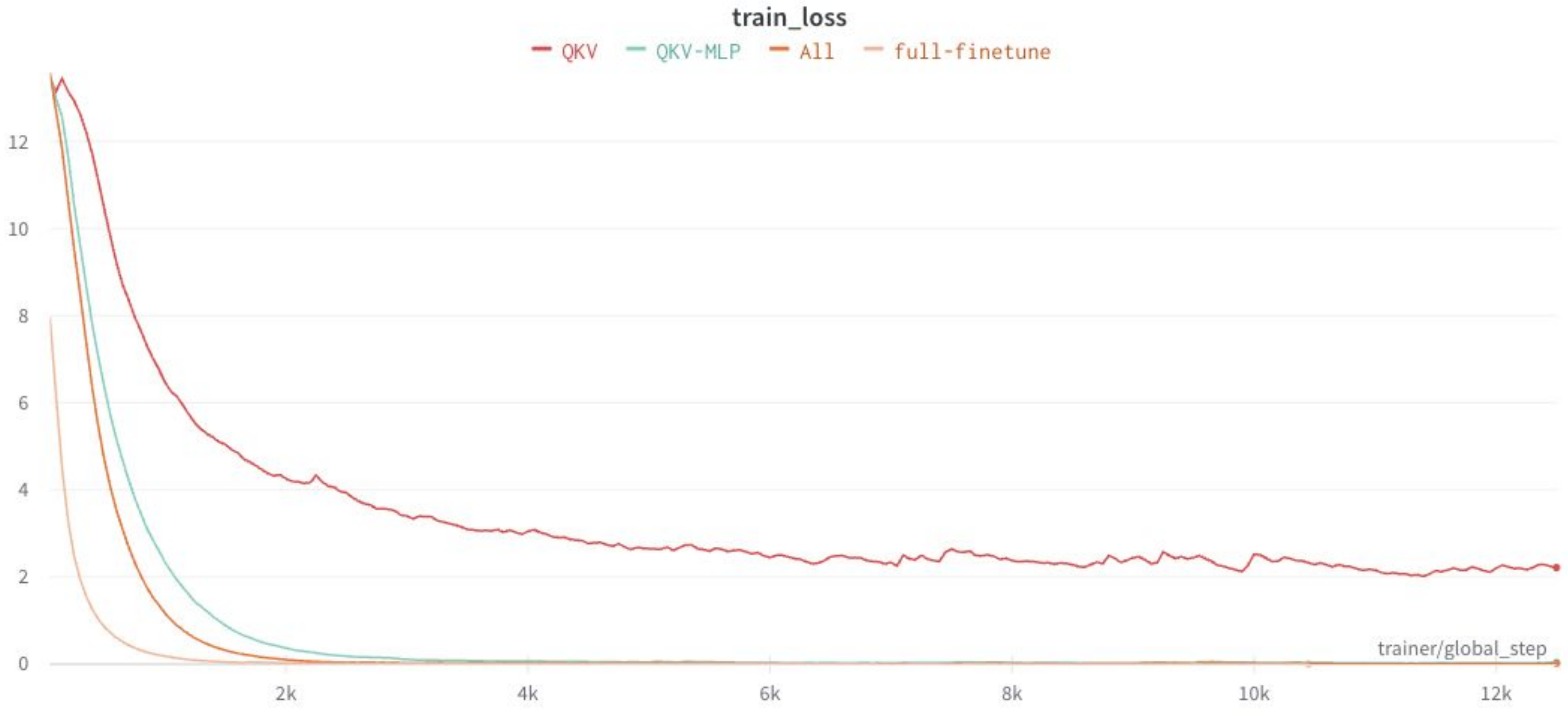


onfido

Accuracy

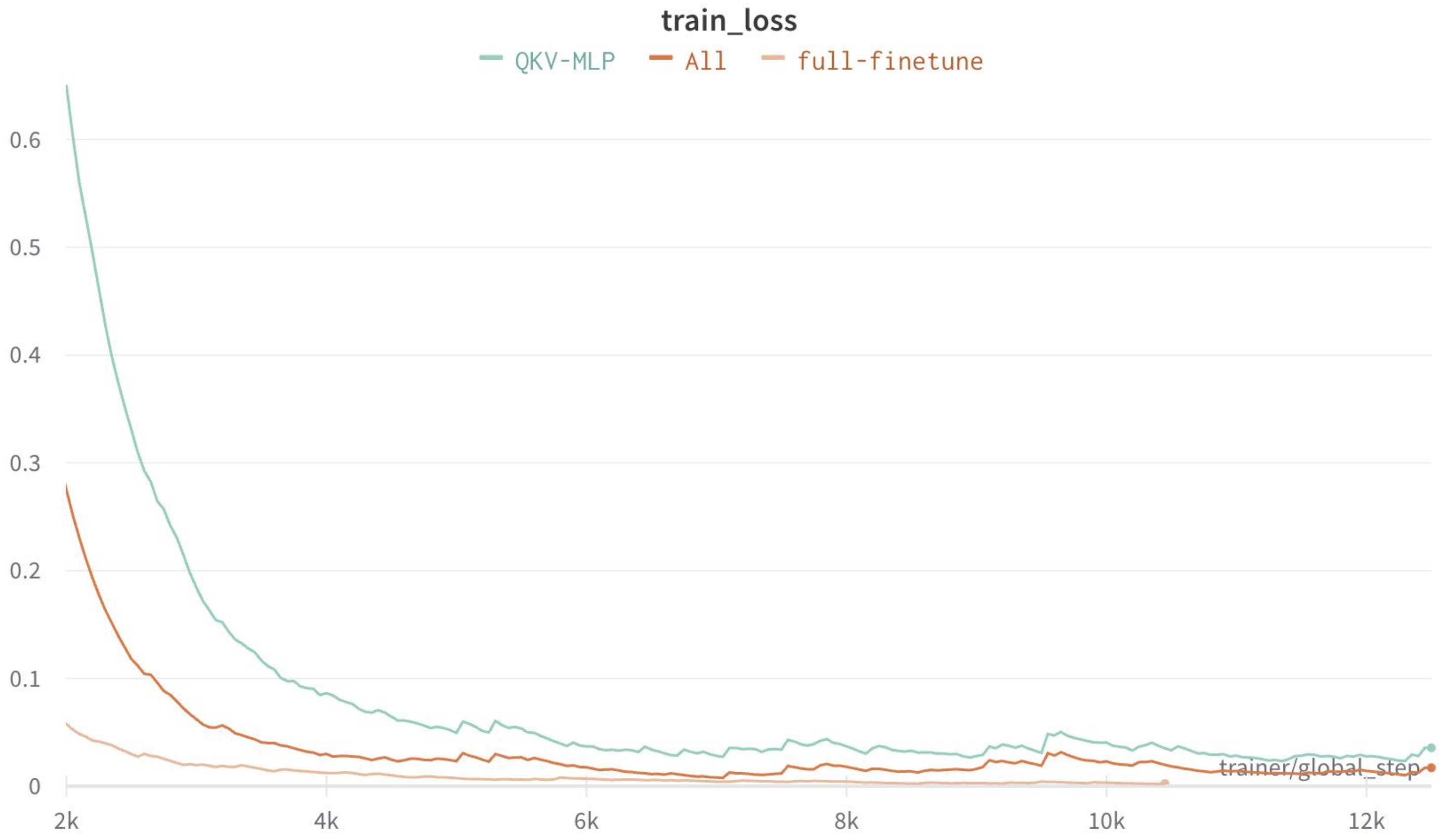
Initial results

adapt everywhere?



Initial results

adapt everywhere?



Initial results

document verification task evaluation

Document Type	**r=8**	**r=16**	**r=32**	**r=64**
Document A	-0.59pp	-0.75pp	-0.29pp	-0.38pp
Document B	-3.1pp	-3.02pp	-3.1pp	-3.31pp
Document C	-40.71pp	-41.21pp	-37.54pp	-35.62pp
Document D	-2.32pp	-1.77pp	-1.92pp	-1.73pp
Document E	-4.36pp	-3.82	-3.38pp	-3.67pp

Initial results

document verification task evaluation

Document Type	r=8	r=16	r=32	r=64
Document A	-0.59pp	-0.75pp	-0.29pp	-0.38pp
Document B	-3.1pp	-3.02pp	-3.1pp	-3.31pp
Document C	-40.71pp	-41.21pp	-37.54pp	-35.62pp
Document D	-2.32pp	-1.77pp	-1.92pp	-1.73pp
Document E	-4.36pp	-3.82	-3.38pp	-3.67pp

Initial results

Using r=64 & adapting everything

< 4.4pp

accuracy regression

Final results

Longer is better

Document Type	10 epochs	20 epochs	35 epochs
Document A	-0.38pp	-0.17pp	+0.3pp
Document B	-3.31pp	-2.02pp	-0.65pp
Document C	-35.62pp	-3.54pp	-1.84pp
Document D	-1.73pp	-1.3pp	-0.84pp
Document E	-3.67pp	-2.35pp	-0.88pp

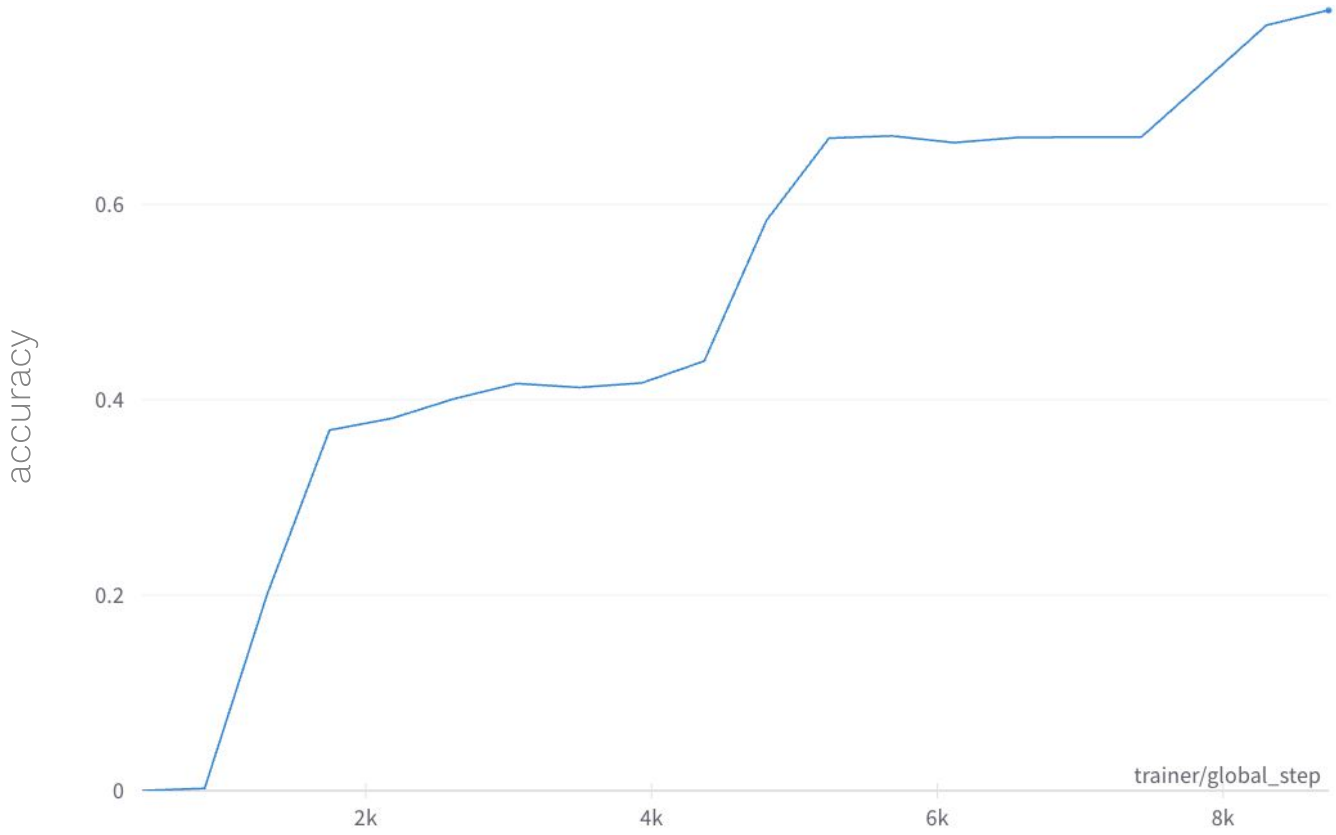
Final results

Longer is better

Document Type	10 epochs	20 epochs	35 epochs
Document A	-0.38pp	-0.17pp	+0.3pp
Document B	-3.31pp	-2.02pp	-0.65pp
Document C	-35.62pp	-3.54pp	-1.84pp
Document D	-1.73pp	-1.3pp	-0.84pp
Document E	-3.67pp	-2.35pp	-0.88pp

The Chart

for Document C



Final results

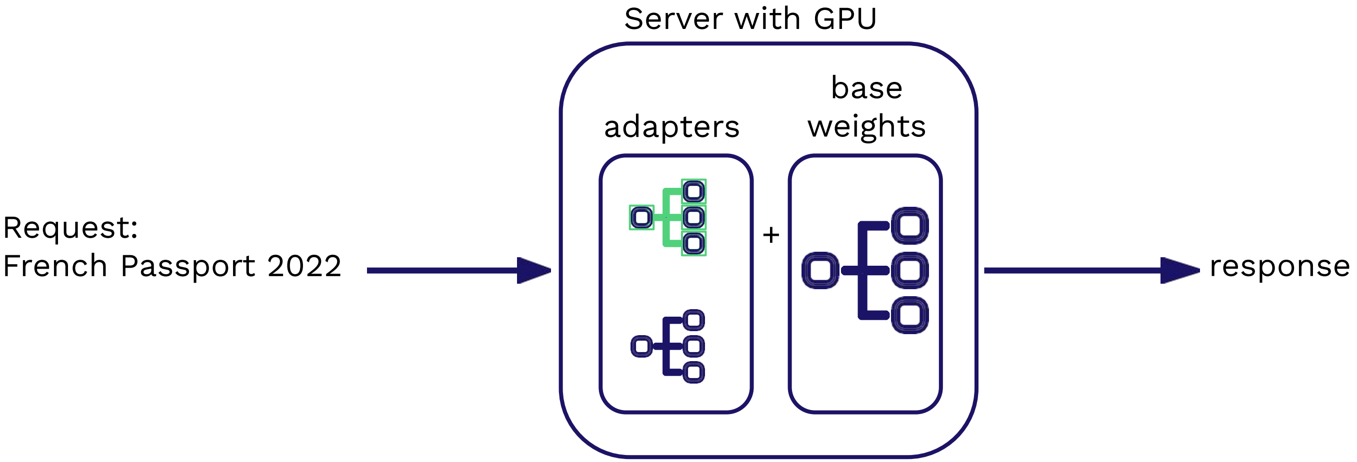
Using r=64, adapting everything & training for longer

< 1pp

accuracy regression

Latency

Remember this?



Initial results

Using local machine & server-side tracing

+80%

latency increase

+100sMb

buffer memory

Initial results

Using local machine & server-side tracing

+80%

latency increase

+100sMb

buffer memory

```
(encoder): ViTEncoder(
  (layer): ModuleList(
    (0-11): 12 x ViTLayer(
      (attention): ViTAttention(
        (attention): ViTSelfAttention(
          (query): Linear(
            in_features=768, out_features=768, bias=True
            (lora_dropout): ModuleDict(
              (default): Dropout(p=0.1, inplace=False)
            )
            (lora_A): ModuleDict(
              (default): Linear(in_features=768, out_features=16, bias=False)
            )
            (lora_B): ModuleDict(
              (default): Linear(in_features=16, out_features=768, bias=False)
            )
            (lora_embedding_A): ParameterDict()
            (lora_embedding_B): ParameterDict()
          )
```

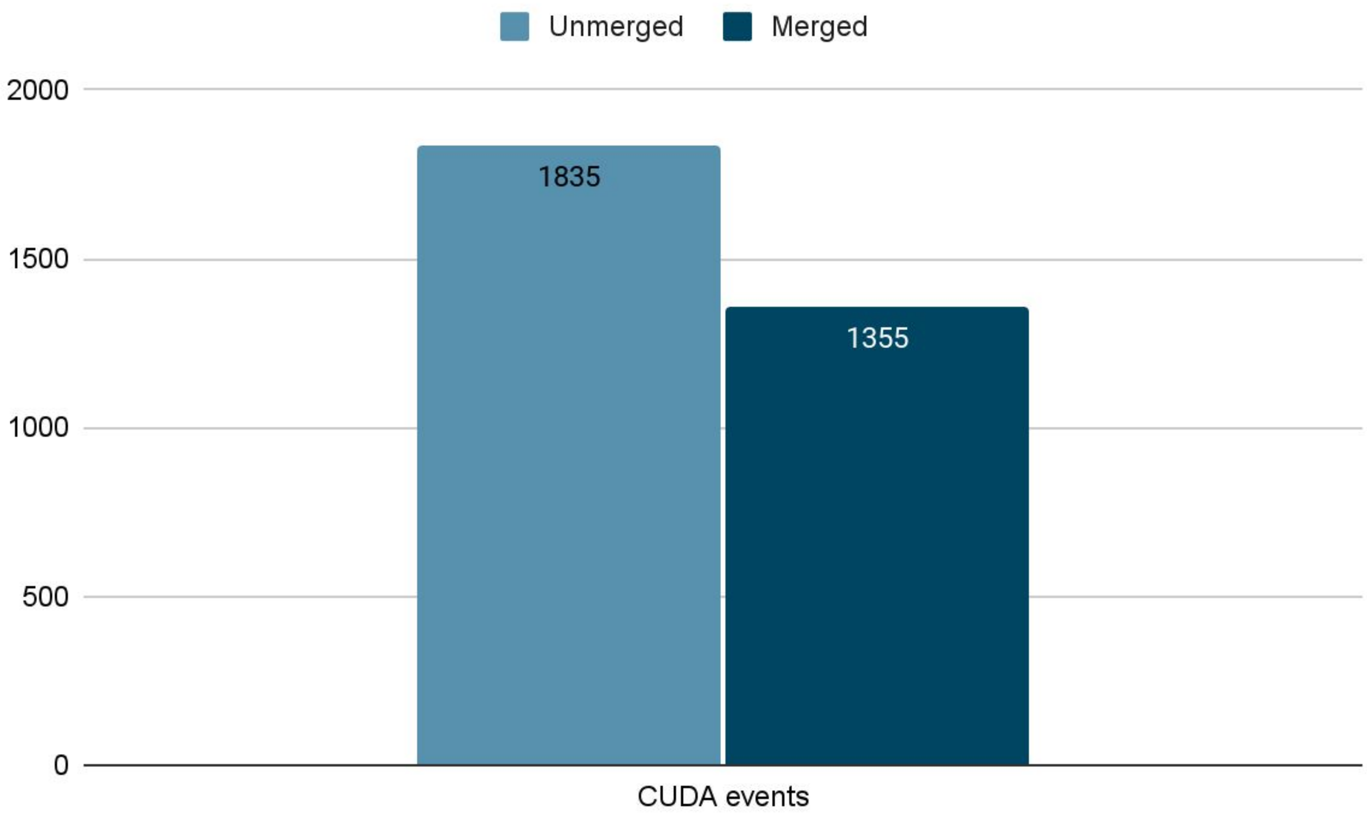
https://github.com/lucapericlp/lora-latencies/blob/master/LoRA_adapter_merging.ipynb

Merging

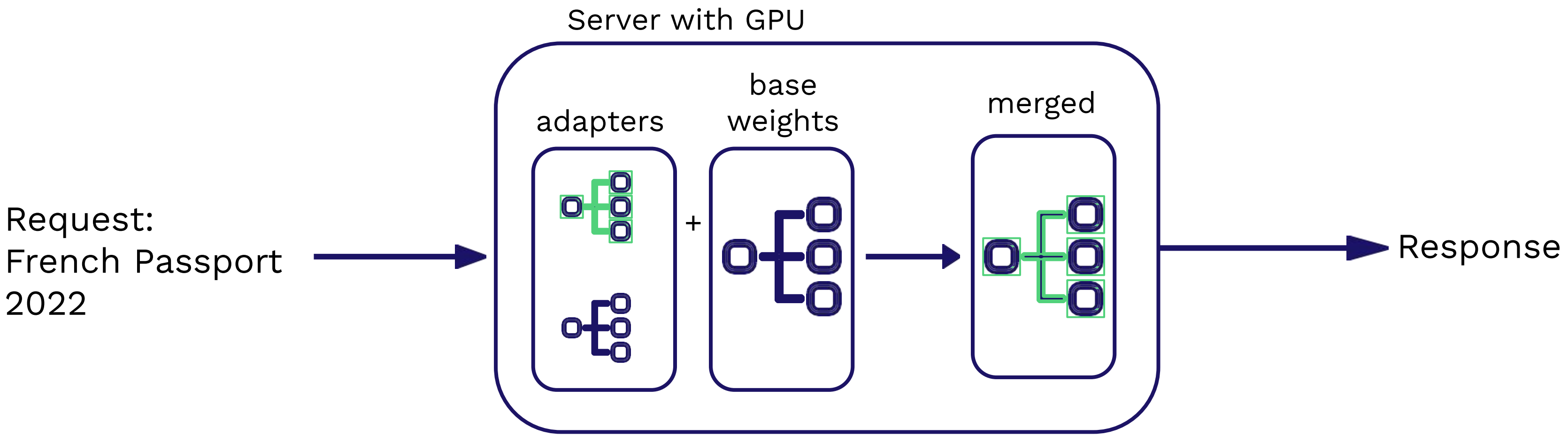
https://huggingface.co/docs/peft/conceptual_guides/lora#merge-lora-weights-into-the-base-model

```
[65] q_one = inference_model.base_model.model.vit.encoder.layer[0].attention.attention.query
     lora_a_linear = q_one.lora_A["default"].weight
     lora_b_linear = q_one.lora_B["default"].weight
     lora_a_linear.shape, lora_b_linear.shape

     (torch.Size([16, 768]), torch.Size([768, 16]))

[66] delta_weights = lora_b_linear @ lora_a_linear
     delta_weights.shape

     torch.Size([768, 768])

[67] merged_q_one = q_one.weight.data + delta_weights
     merged_q_one.shape

     torch.Size([768, 768])
```

Inspection



What it really looks like



Toggling adapters

```
def run(self, request: Request):
  if self.model.active_adapter != request.adapter:
      self.model.unmerge_adapter()
      self.model.set_adapter(request.adapter)
      self.model.merge_adapter()
  result = self.model.inference(request.data)
  return Response(result)
```

Revisited results

Using local machine & server-side tracing

+1.5%

latency increase

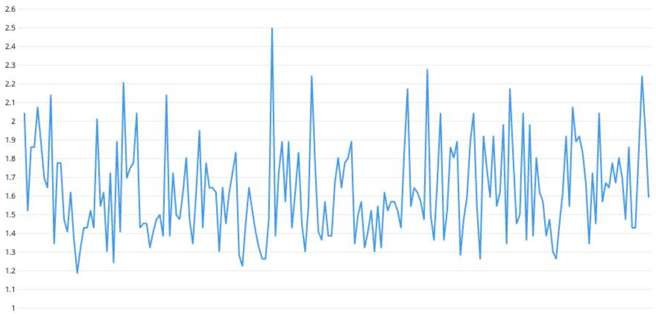
+0Mb

buffer memory

Load response

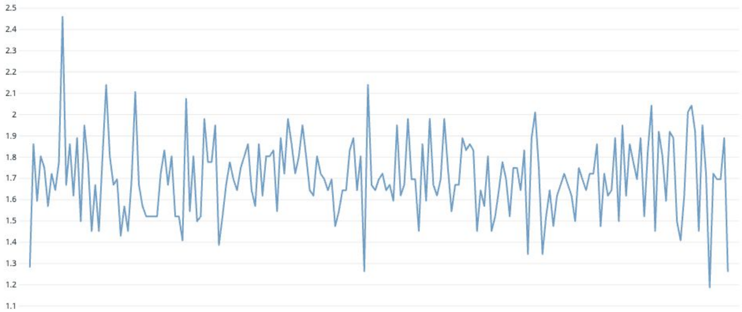
in production, targeting 5 doc types

fully finetuned model



p95: 1.69s

LoRA



p95: 1.77s (+0.08s)

Results

onfido

-97.5%
hardware reqs

~80ms
latency increase

< 1.0pp
accuracy decrease

Agenda

onfido

1 **Intro**
Who are we?

2 **Identity Verification**
Why automate it?

3 **High Performance at Scale**
How do we get there?

4 **Actionable LoRA**
How can you use it?

Takeaways

- Adapt everything
- Higher rank is better (w/ diminishing returns)
- `peft` is great
- Increase batch size, train for longer
- Alpha parameter YMMV
- Merge & unmerge adapters at request time

onfido

Software Engineer - (Machine Learning)

APPLY FOR THIS JOB

Remote - United Kingdom

Technology – Engineering / Full-Time / Remote

More resources

- https://lightning.ai/pages/community/lora-insights/
- https://magazine.sebastianraschka.com/p/practical-tips-for-finetuning-llms
- https://www.anyscale.com/blog/fine-tuning-llms-lora-or-full-parameter-an-in-depth-analysis-with-llama-2
- https://arxiv.org/abs/2305.14314
- Empirical Analysis of the Strengths & Weaknesses of PEFT Techniques for LLMs
- A Comparative Study between Full-Parameter and LoRA-based Fine-Tuning

More resources

Luca Perić
@lp_peric

Over 3.5 weeks, I worked on serving hundreds of transformer-based vision models at scale via LoRA @Onfido with the aim of preserving as much accuracy as possible.

These are 5 resources that I found particularly helpful 🧵

4:15 PM · Nov 28, 2023

Takeaways

- Adapt everything
- Higher rank is better (w/ diminishing returns)
- `peft` is great
- Increase batch size, train for longer
- Alpha parameter YMMV
- Merge & unmerge adapters at request time