AI on a microbudget

Methods of machine learning miniaturization

Katharina Rasch & Christian Staudt

https://github.com/ai-dojo/microbudget

About us

Christian Staudt

Freelance Data Scientist

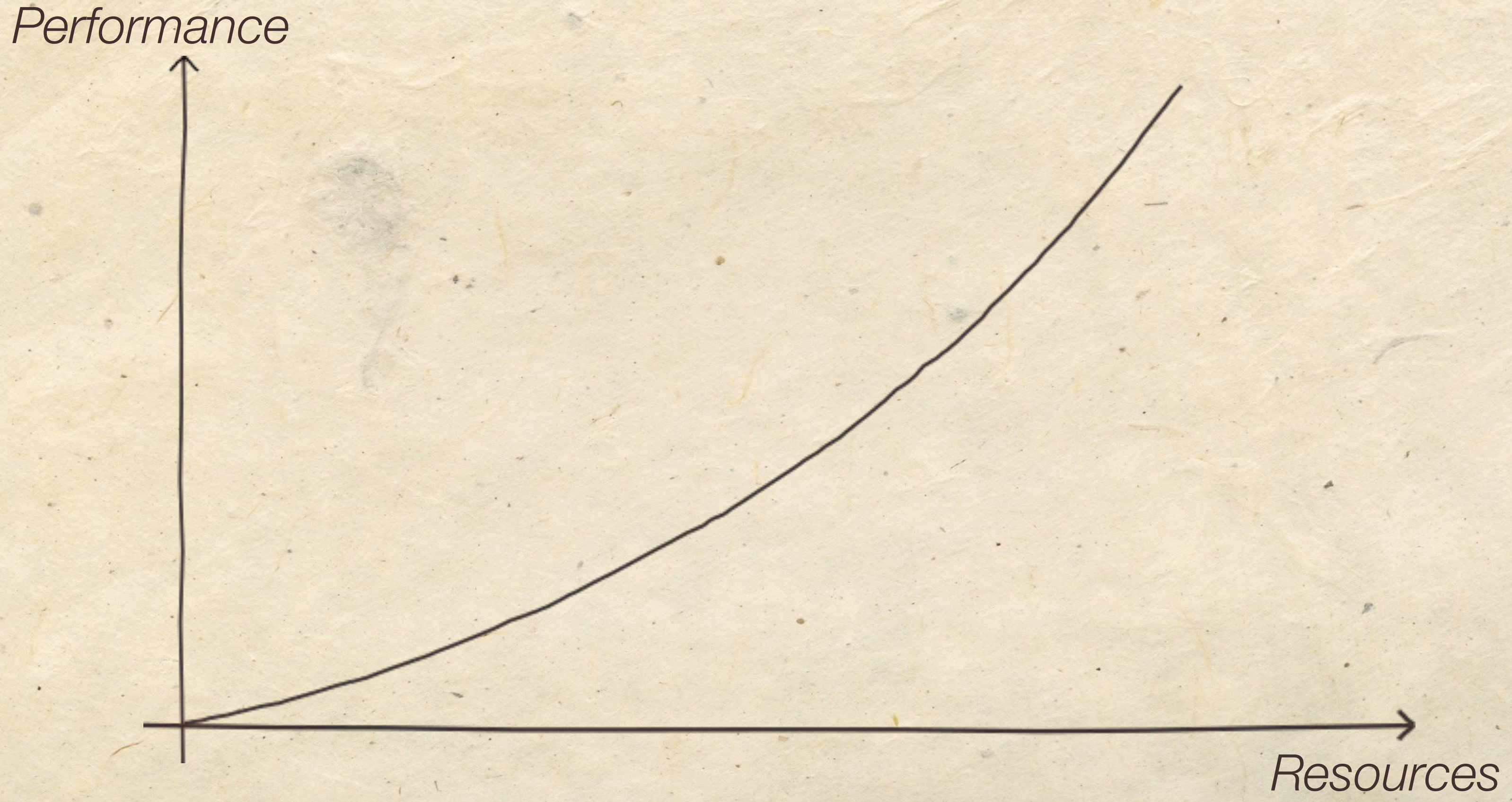
https://clstaudt.me

Katharina Rasch

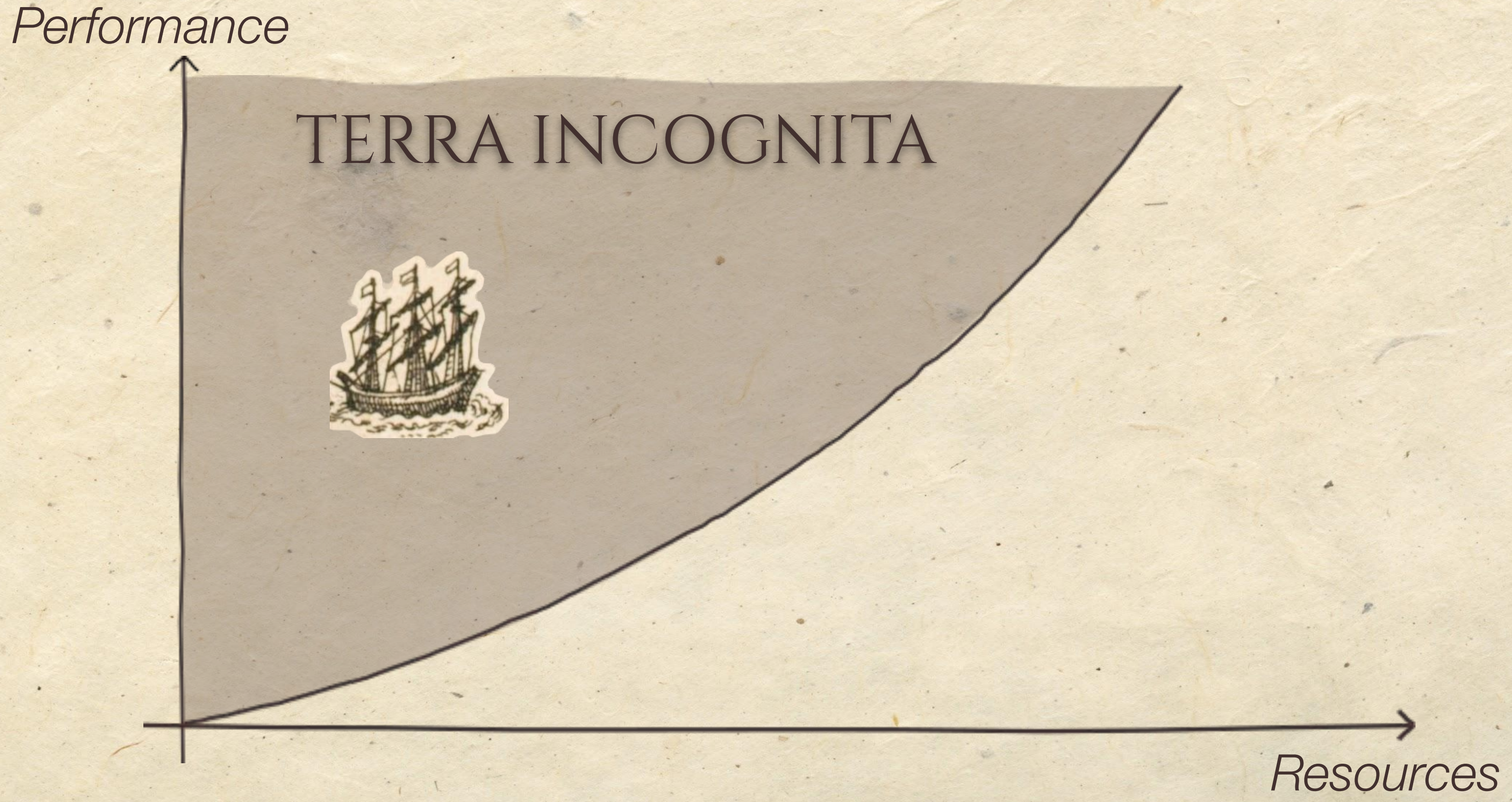
Freelance Data Scientist

https://krasch.io/

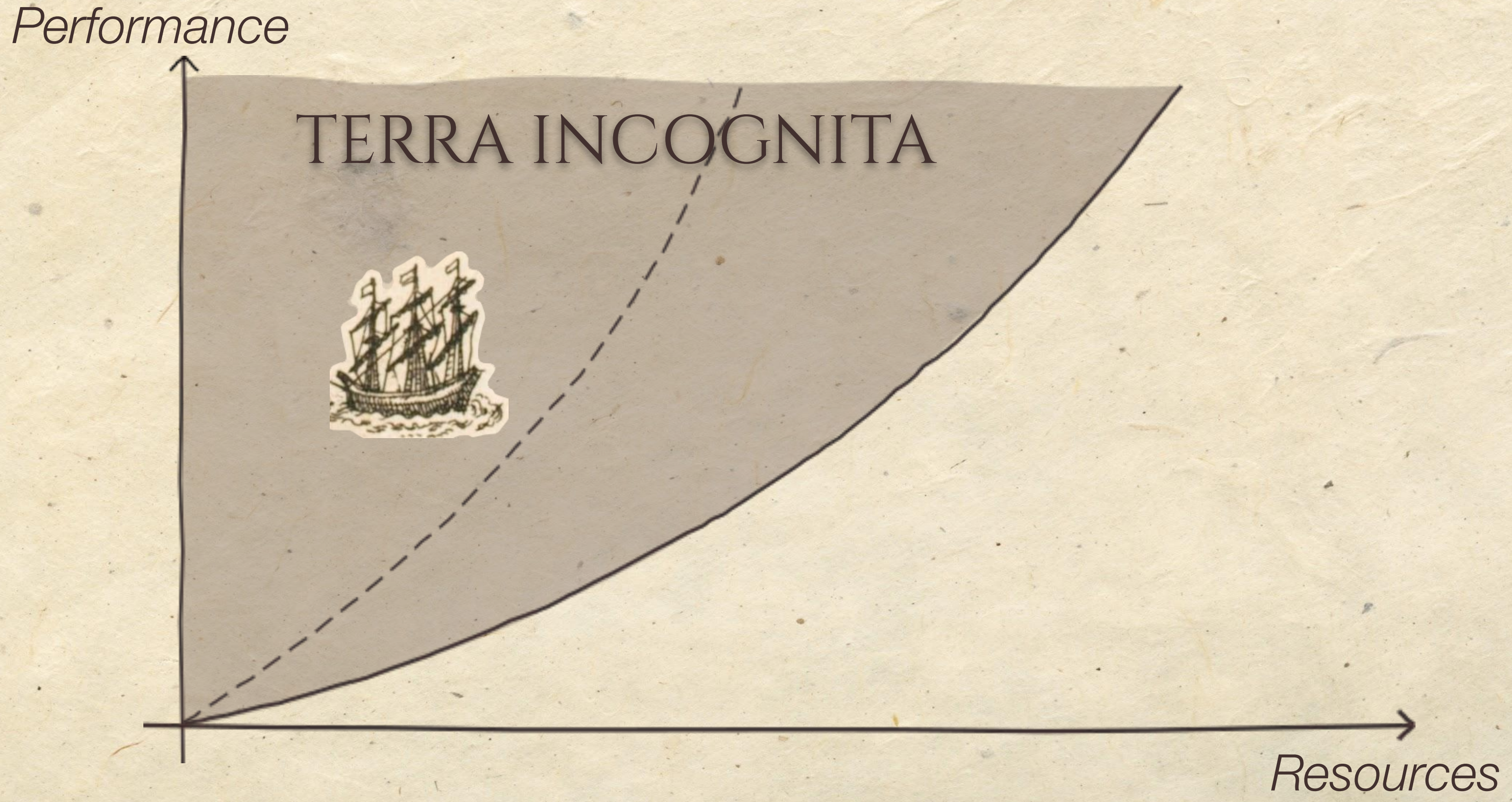
Scaling is all you need?



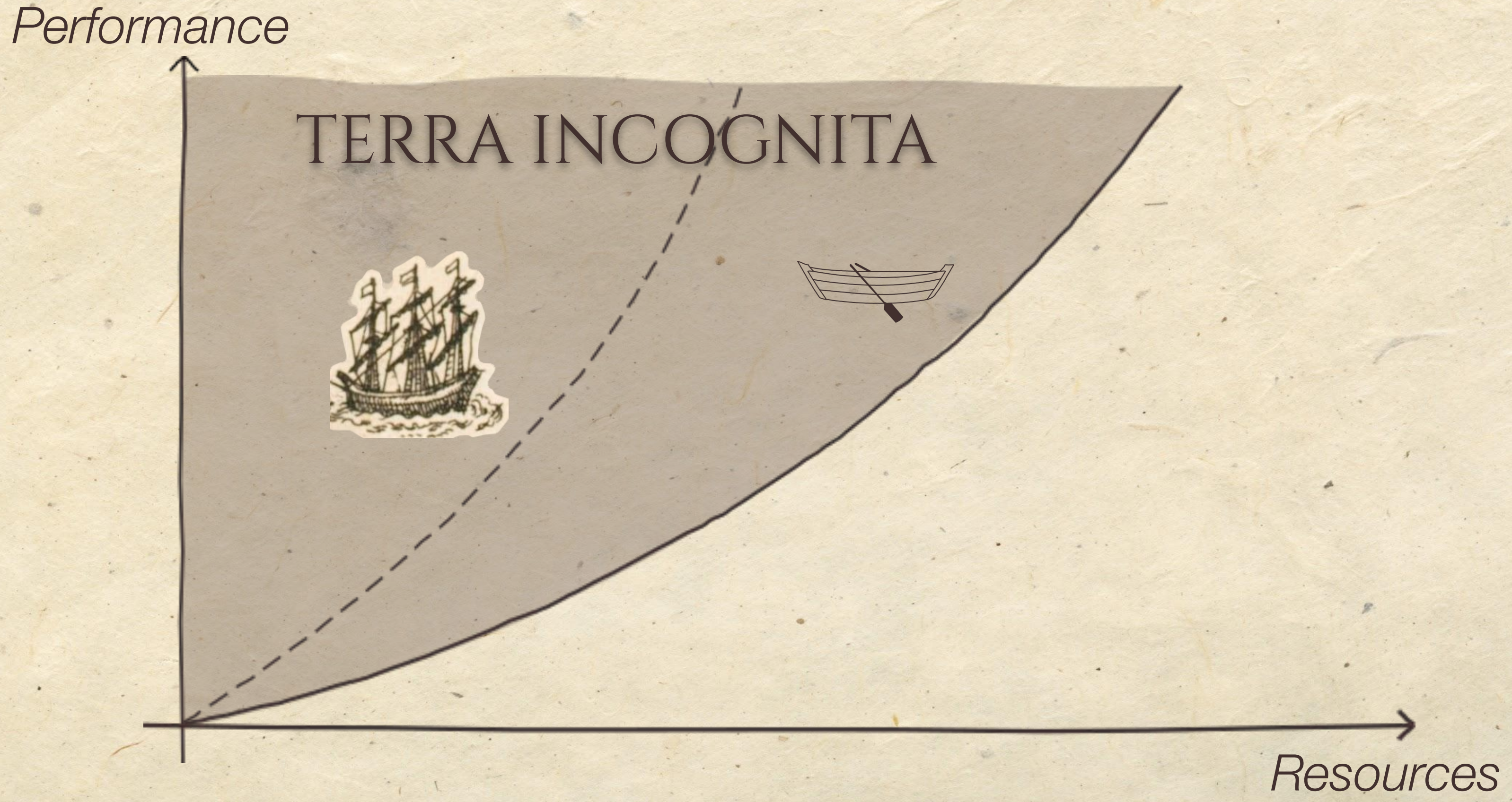
Scaling is all you need?



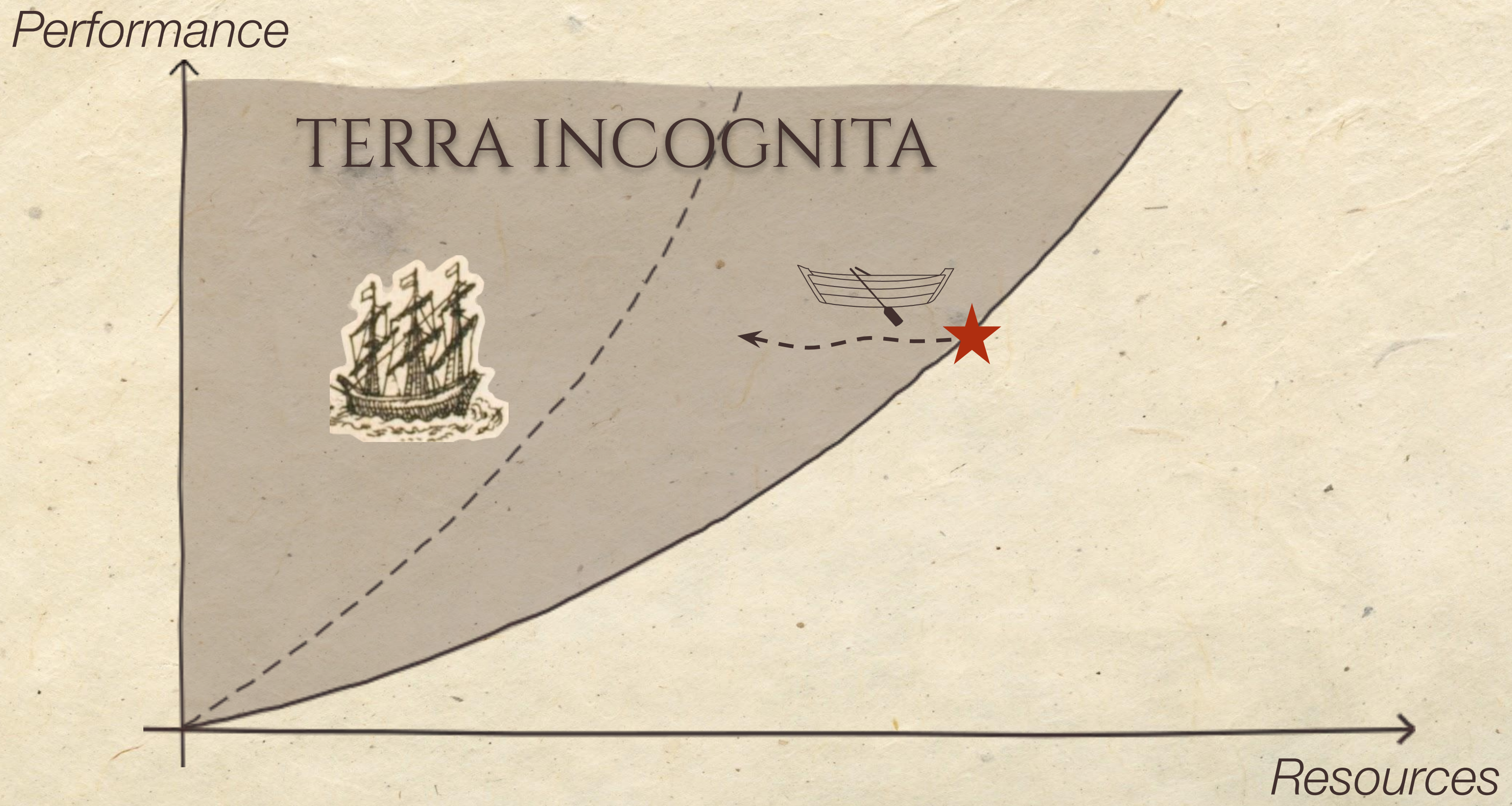
Scaling is all you need?



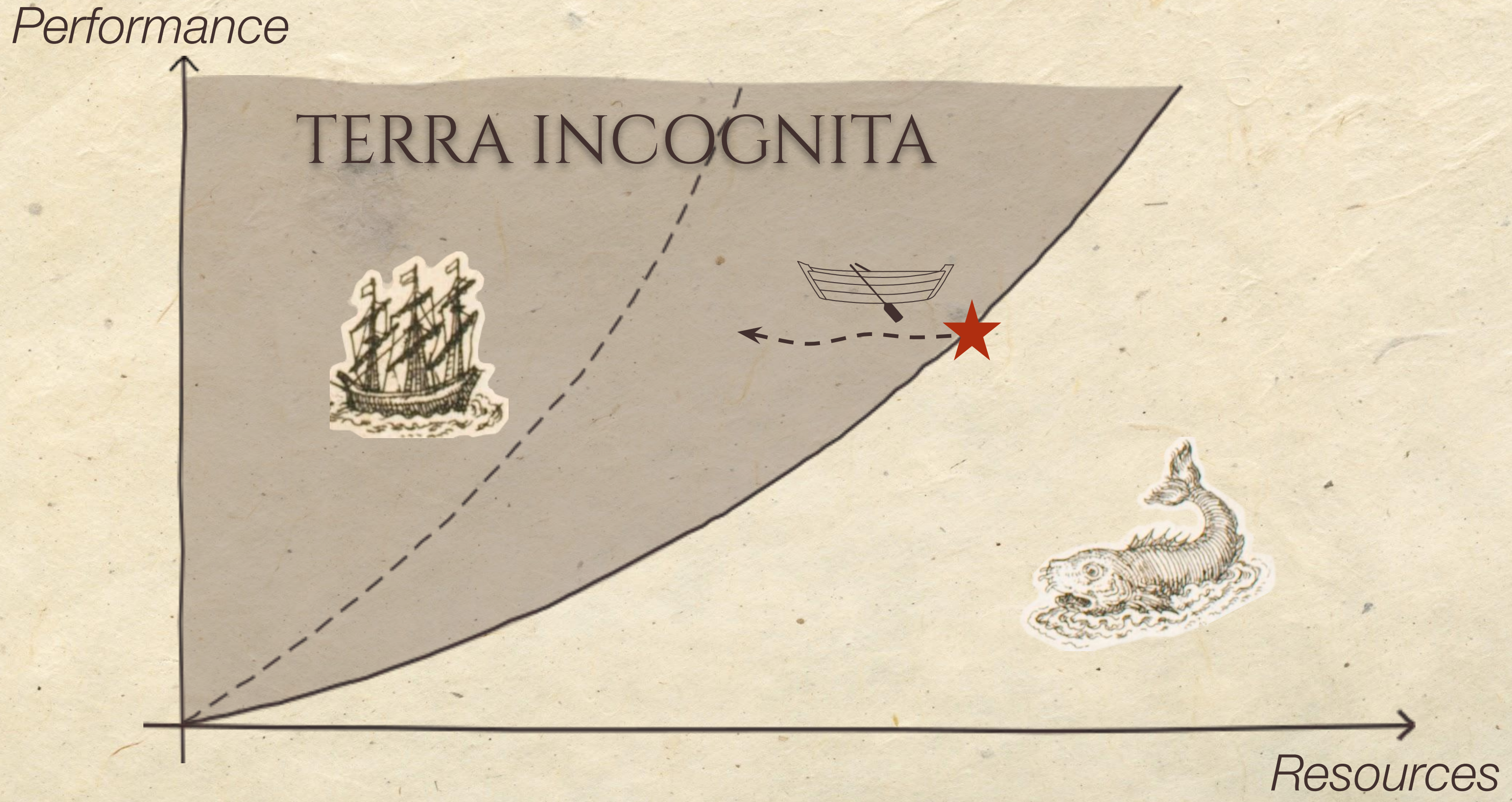
Scaling is all you need?



Scaling is all you need?



Scaling is all you need?



Plan for today

- 3 microbudget methods for adapting existing pre-trained models to your needs:
  - Transfer learning
  - Distillation
  - Quantization

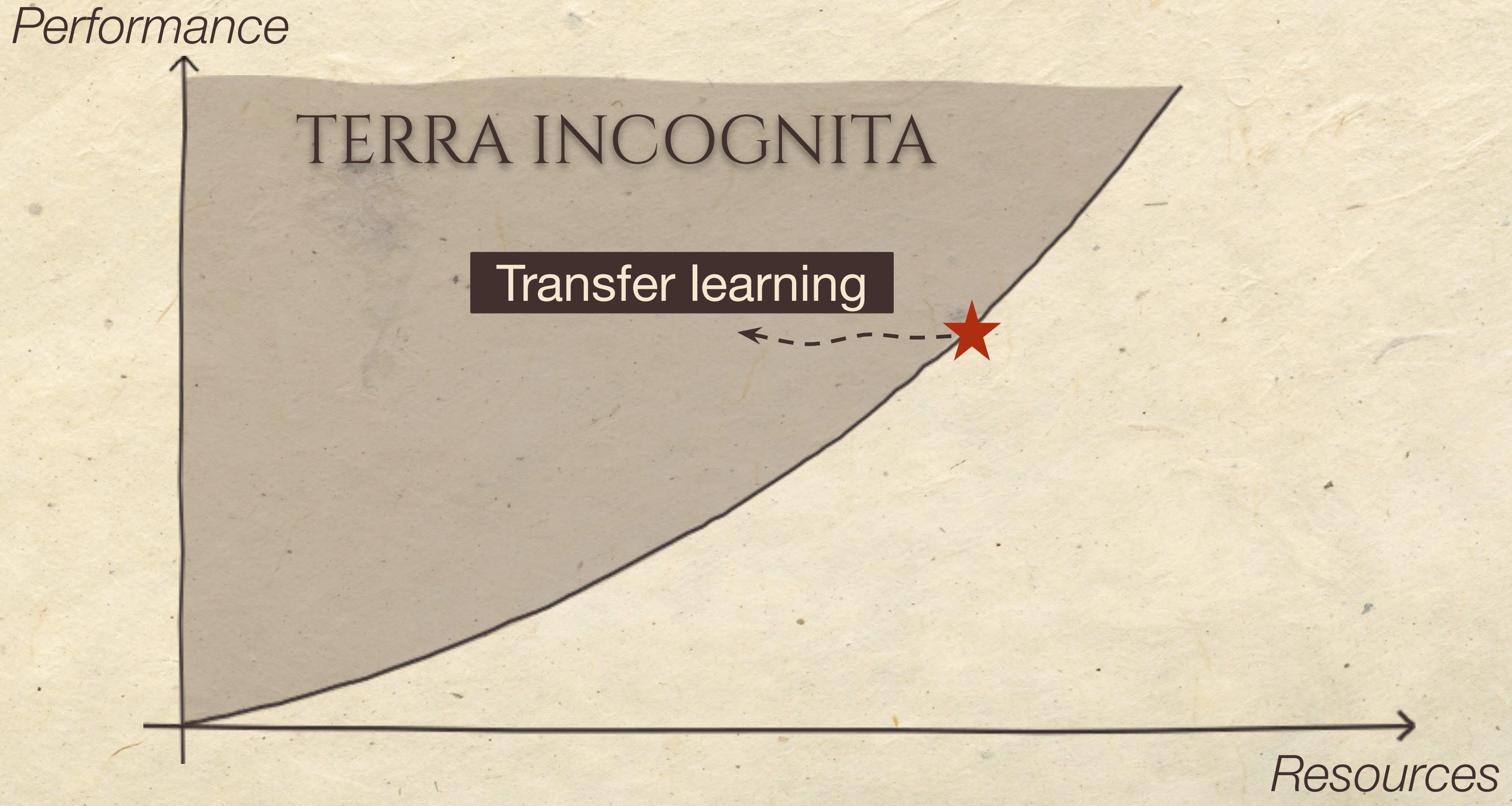
- Microbudget = small team, a few GPUs, small datasets

Companion repository

https://github.com/ai-dojo/microbudget

- A collection of example notebooks for microbudget ML methods
    - 📙 Transfer learning for building a custom Image Classifier
    - 📙 Faster Speech Transcription through Model Distillation
    - 📙 Running a Large Language Model with Different Levels of Quantization

Method 1: Transfer learning



Example from our companion repo

- Task: Create a classifier for botanical images

Oxford 102 Flowers dataset

- ca. 8000 images
- ca. 500px height
- ca. 400 MB
- 102 classes

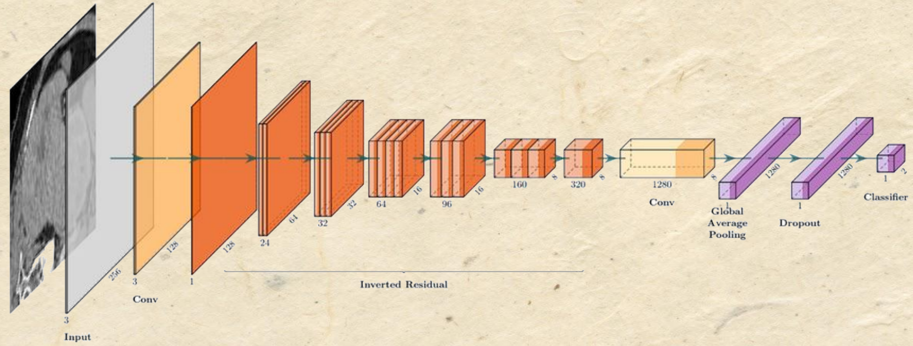
training computer vision model from scratch = straining our microbudget

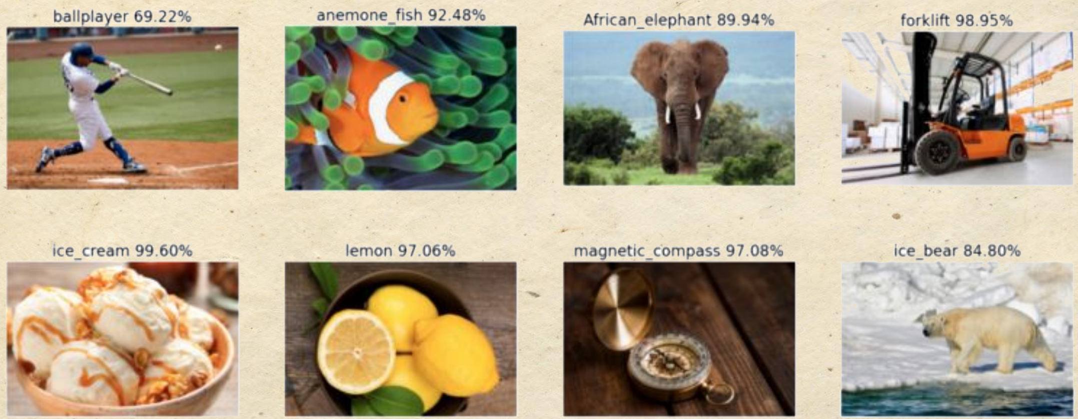
Example from our companion repo

reuse an existing image classifier?

e.g. *MobileNetV2*

- deep CNN (53 layers)
- 3.4 M parameters
- good accuracy on ImageNet





ImageNet dataset

- 1.4 million images
- 1000 classes
- > 150 GB

Basic idea idea of transfer learning

Model pre-trained for Task A on large dataset

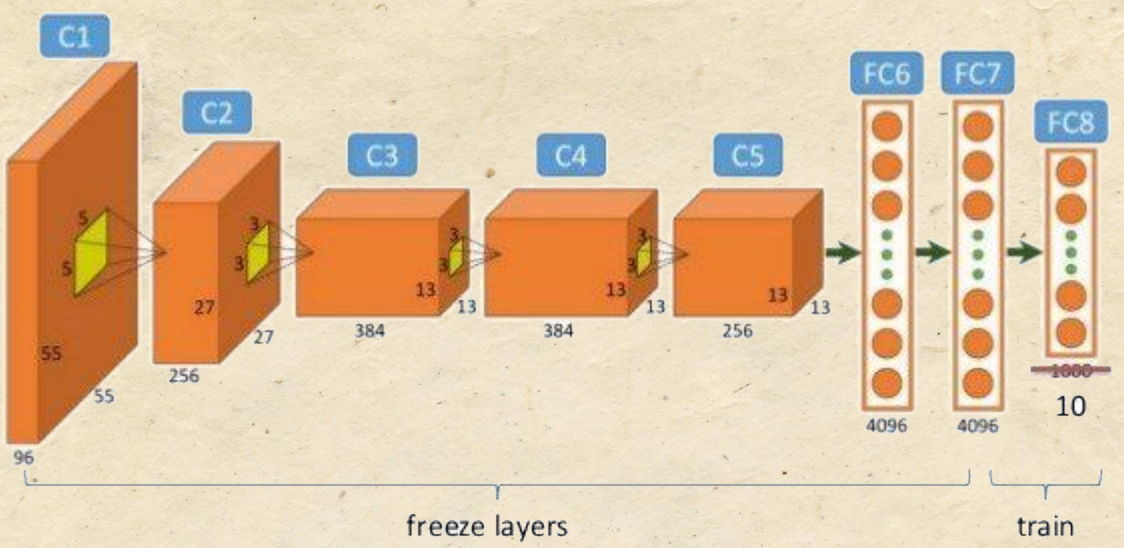
partially retrain on (smaller) dataset for Task B

Model specialised for Task B

📙 Example from our companion repo

reuse convolutional layers (incl. weights) = feature extraction capabilities

Image source: A Study Review: Semantic segmentation with Deep Neural Networks



adapt & retrain for new task

📙 Example from our companion repo

```python
# Load MobileNetV2 without the top layer
base_model = keras.applications.MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    weights="imagenet",
)
```

[7] ✓ 0.3s Python

```python
# Create a new model on top of the output of the base model
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(n_classes, activation='softmax')
])
```

[15] Python

Transfer learning has great library support

Example from our companion repo

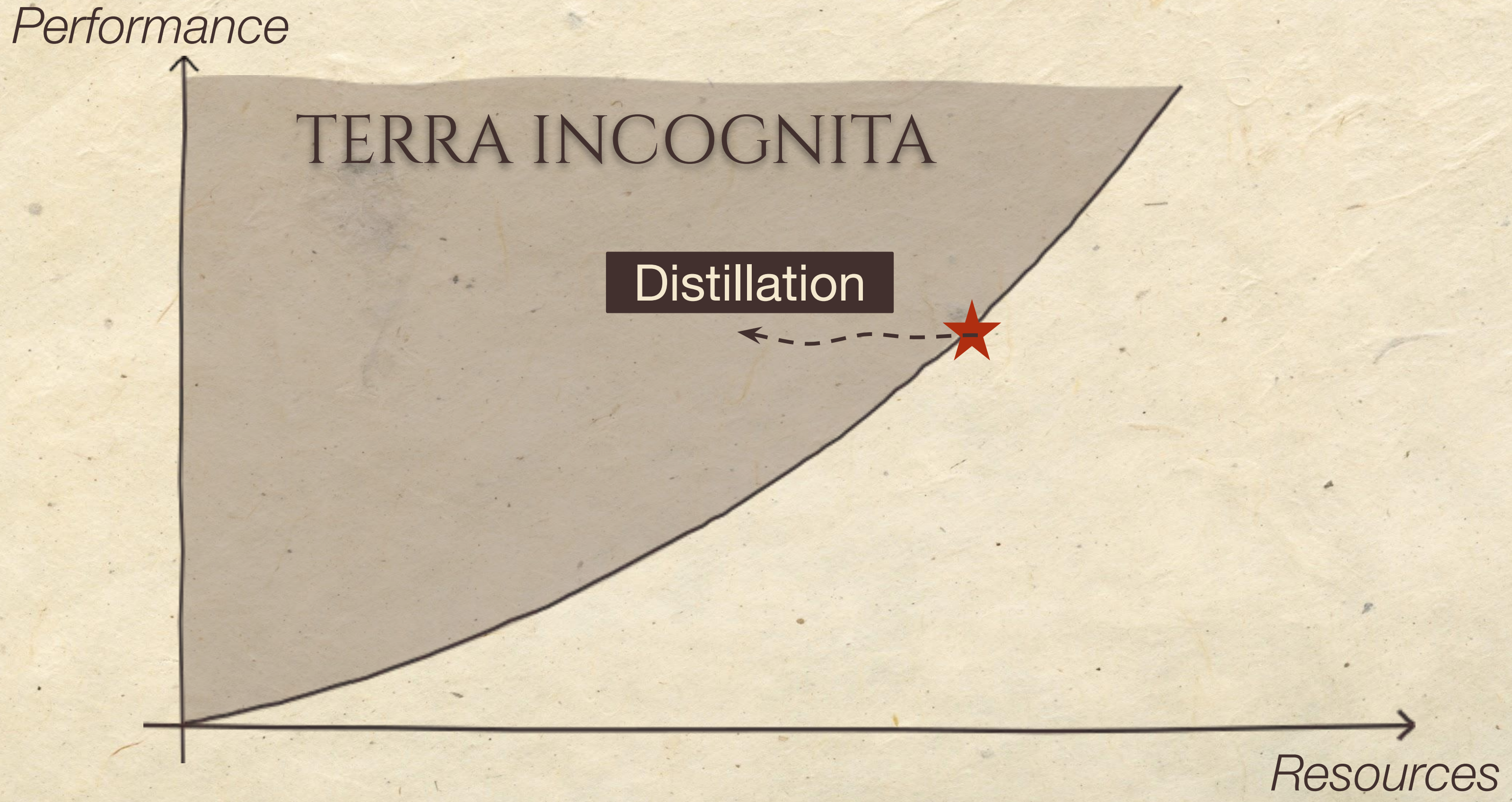
a decent image classifier with minimal engineering & training

Transfer Learning at a glance

	Transfer learning
Model capability	different task
Model size / inference cost	same *
Training data and cost	less *
Development effort	simple

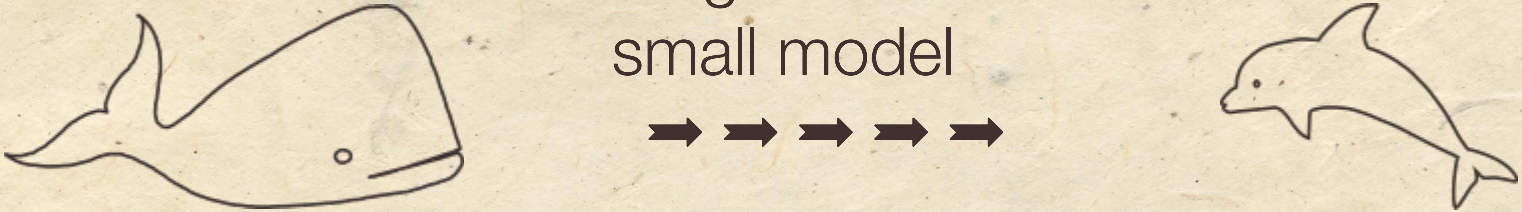
* compared to original model

Method 2: Distillation



Basic idea of model / knowledge distillation

Large model teaches
small model



Large pre-trained teacher model

Small student model

Real-world example: Whisper → Distil-Whisper

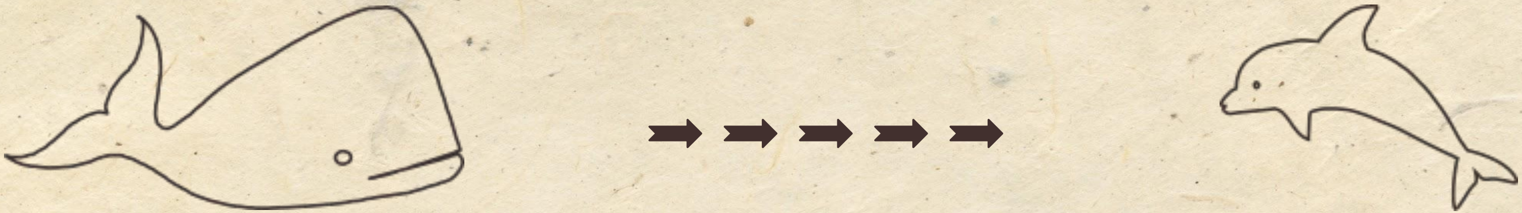
OpenAI Whisper

“Look, penguins!”

trained on 680,000 hours of audio and transcripts

1.55 Billion parameters (for large model)

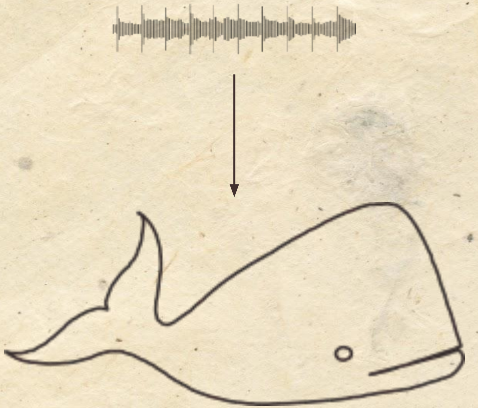
Real-world example: Whisper → Distil-Whisper



OpenAI Whisper

HuggingFace Distil-Whisper

Real-world example: Whisper → Distil-Whisper



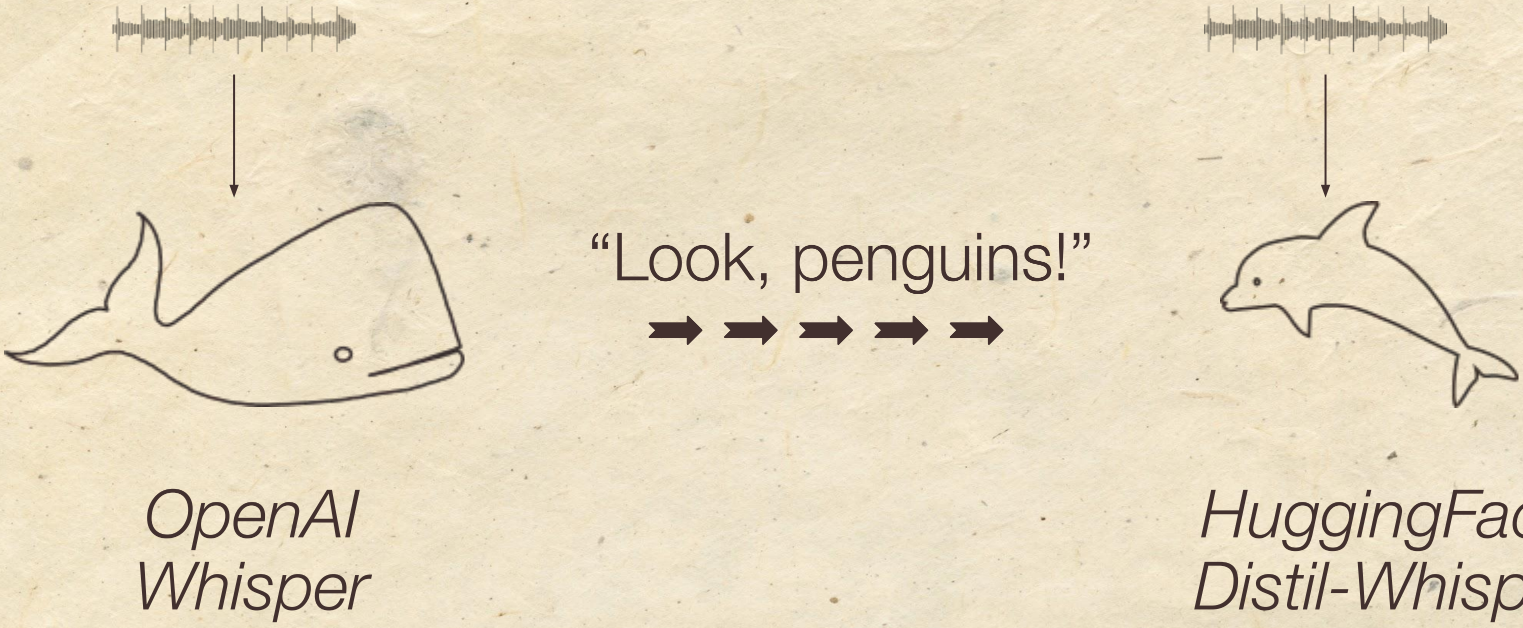
"Look, penguins!"

➡ ➡ ➡ ➡ ➡

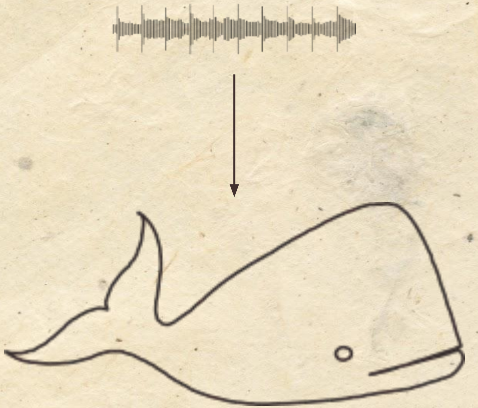
OpenAI Whisper

HuggingFace Distil-Whisper

Real-world example: Whisper → Distil-Whisper

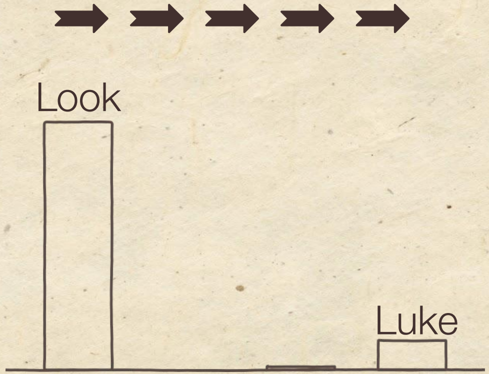


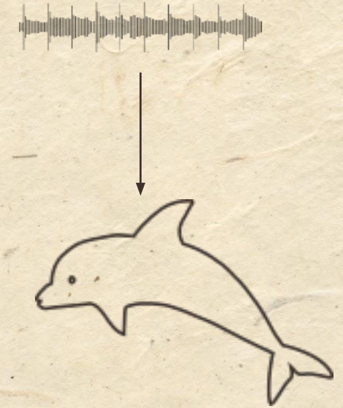
Real-world example: Whisper → Distil-Whisper



OpenAI Whisper

“Look, penguins!”



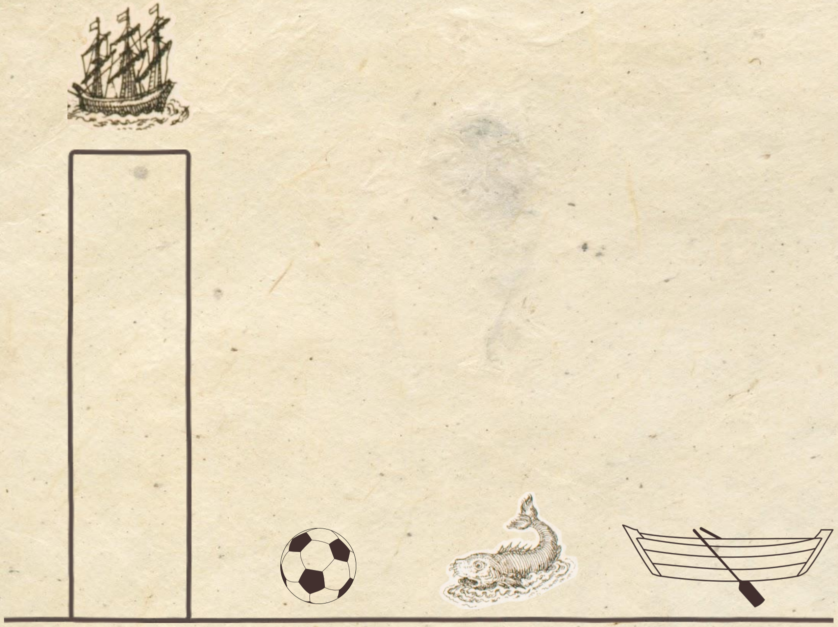


HuggingFace Distil-Whisper

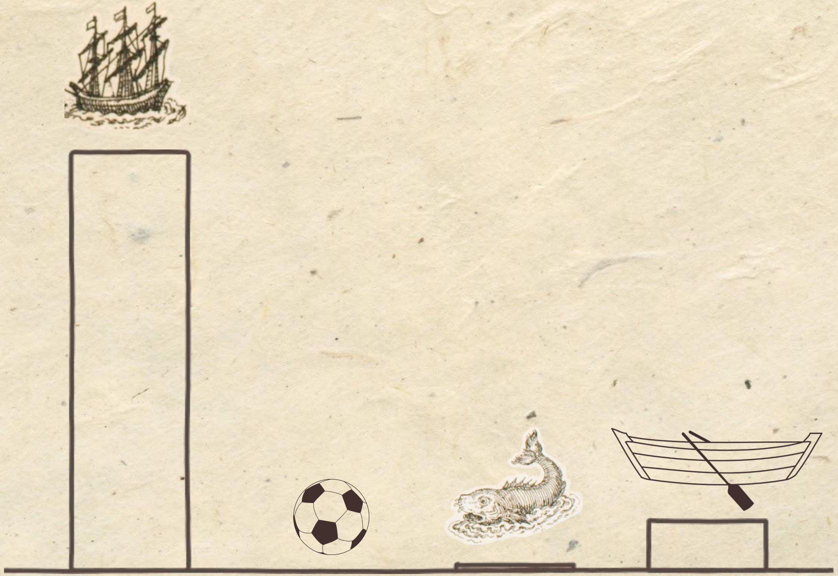
Benefits of distillation

- Less annotated training data needed
- Without distillation, might not be able to train capable small model from scratch, even with full dataset → why?

Our training regime is quite harsh



We train our models with hard labels

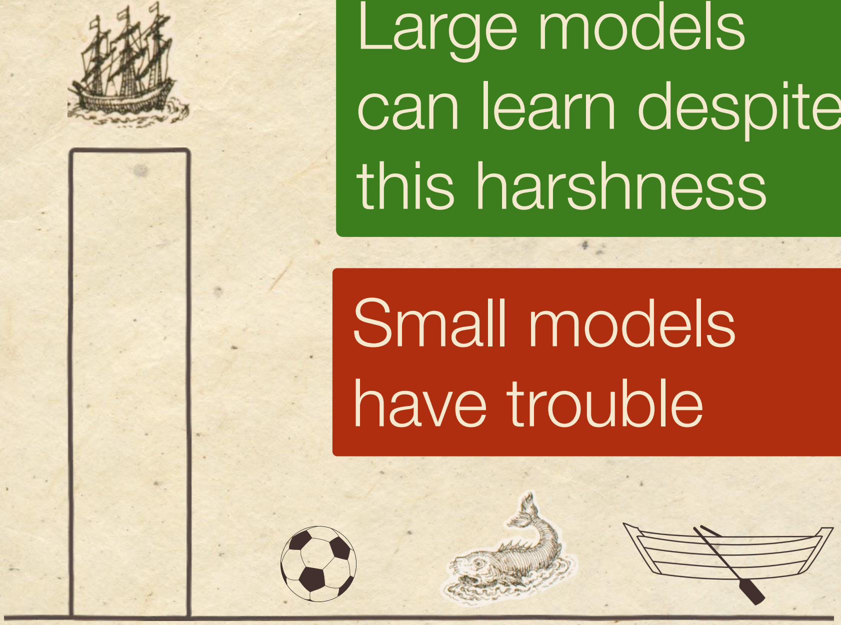


And punish them if they produce soft predictions

Our training regime is quite harsh

Large models can learn despite this harshness

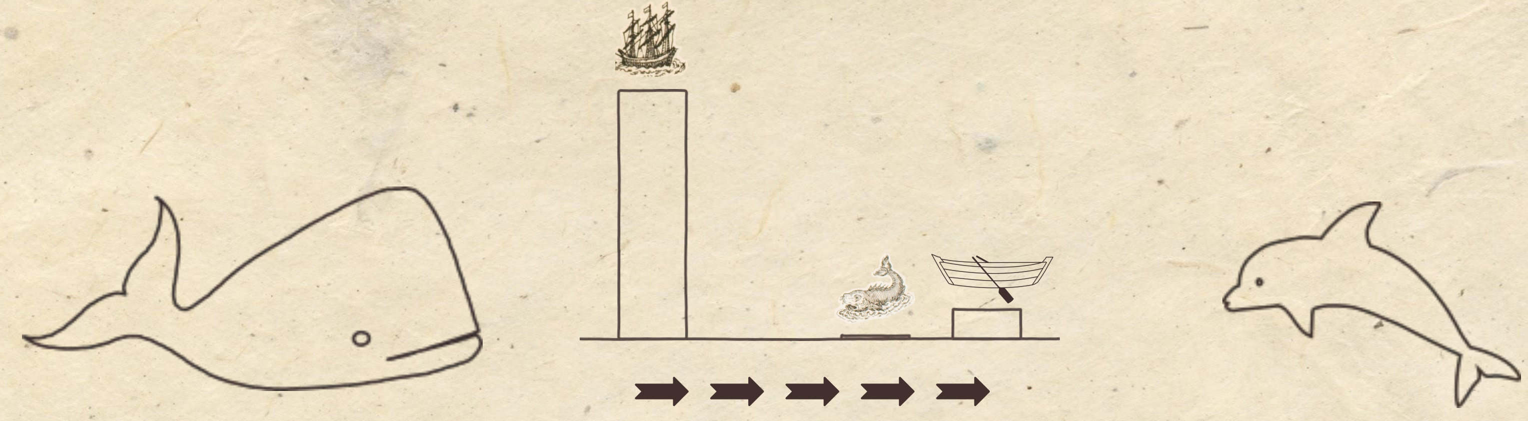
Small models have trouble



We train our models with hard labels

And punish them if they produce soft predictions

Distillation creates a more friendly learning environment



Real-world example: Whisper → Distil-Whisper

Distil-Whisper ends up being

- 6 times faster
- 50% smaller
- within 1% word error rate (WER) of original model

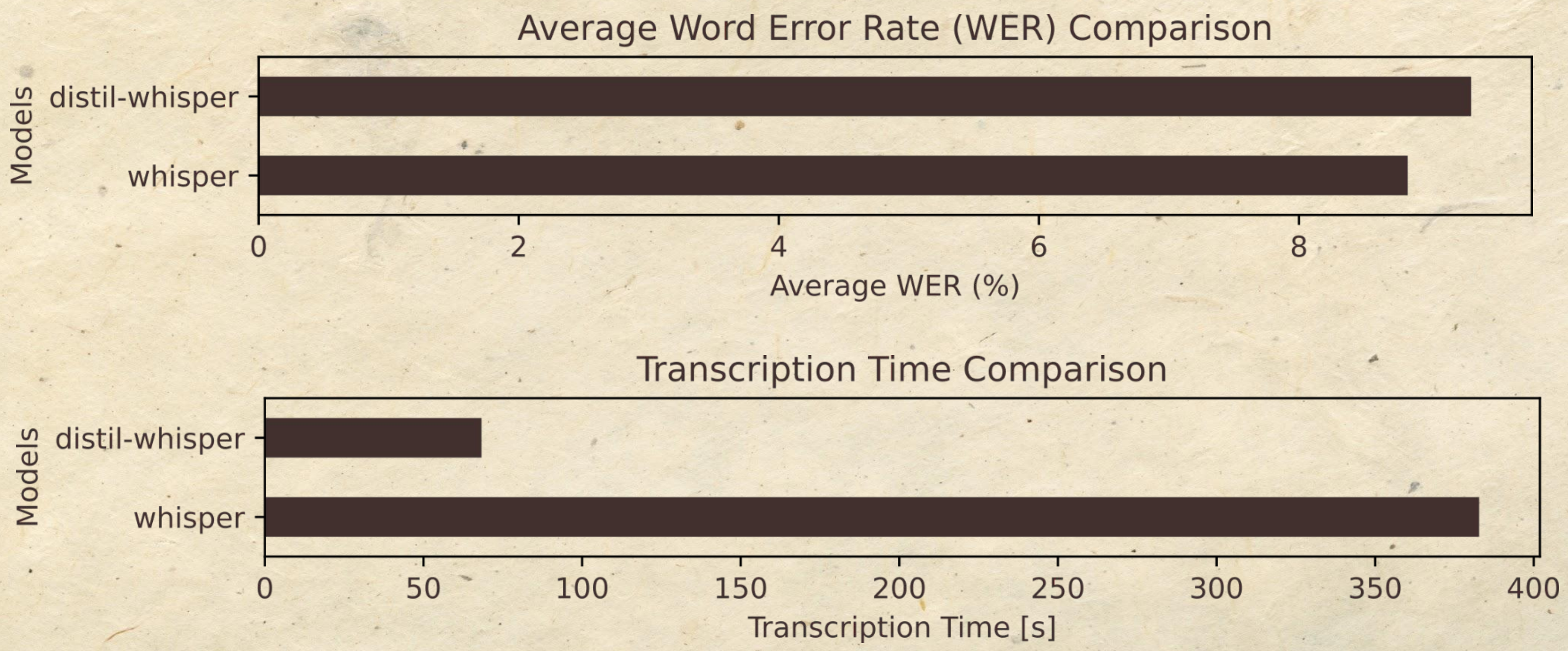
Drop-in replacement!*

*for English

distillation cost? → trained on 14kh of audio instead of 680kh = ca. 2% of original

📙 Example from our companion repo

Transcribe with whisper or distil-whisper: see (and hear) for yourself



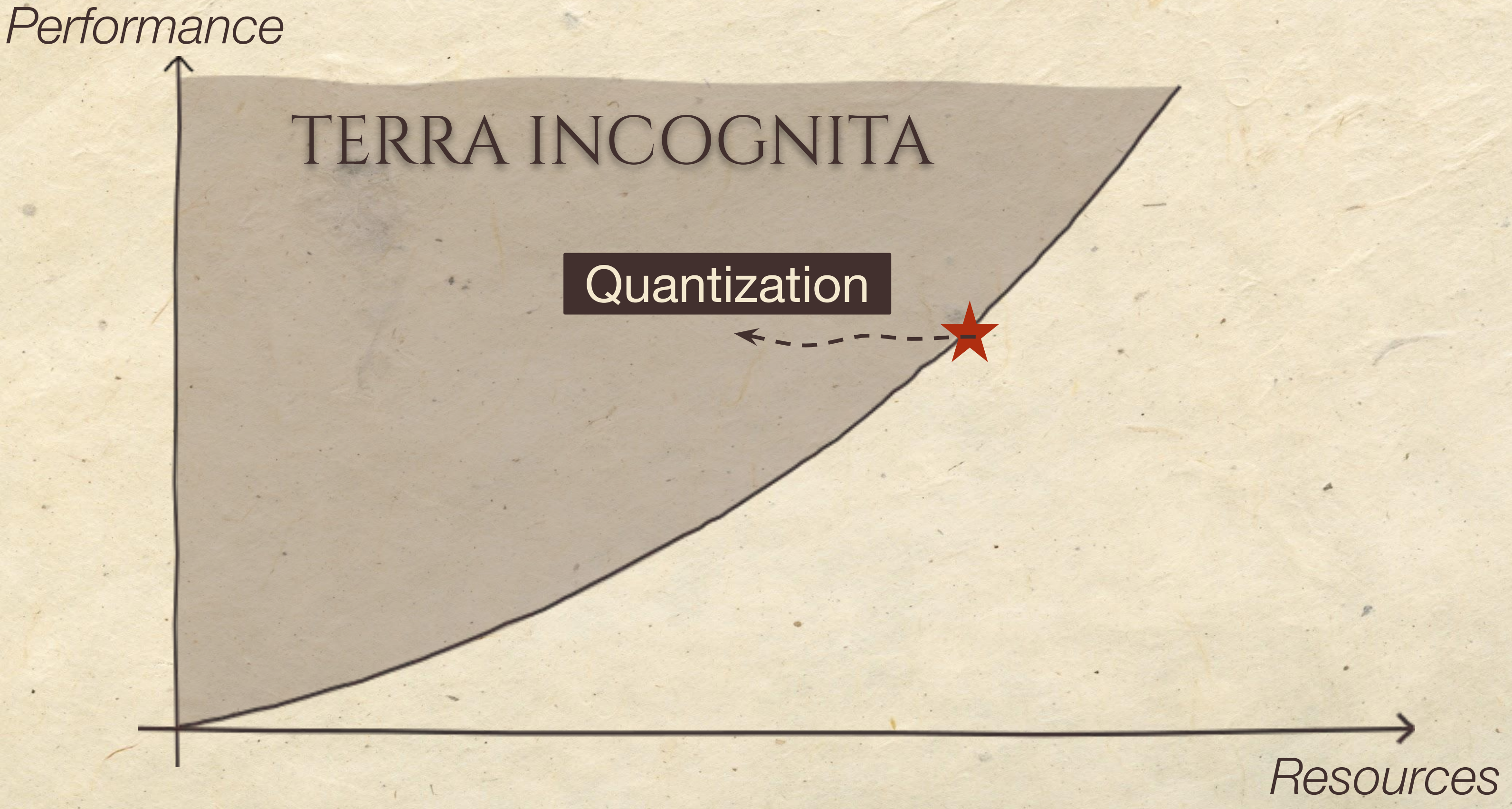
Medium-size model run on CPU, for 32 librivox audio samples

Distillation at a glance

	Transfer learning	Distillation
Model capability	different task	potentially same *
Model size / inference cost	same *	much smaller *
Training data and cost	less *	less *
Development effort	simple	complex

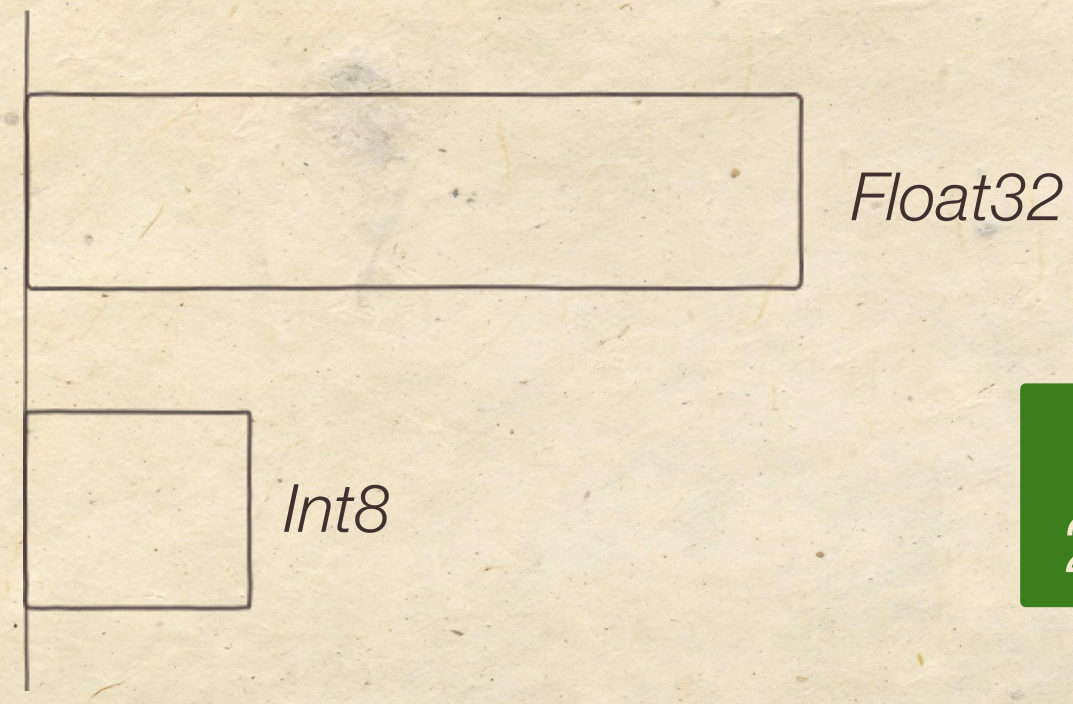
* compared to original model

Method 3: Quantization



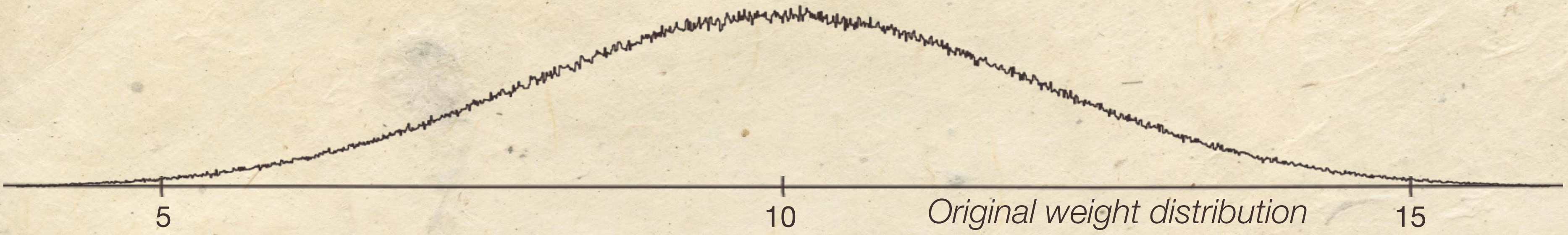
Basic idea of quantization

Do we need full precision weights to represent a model's knowledge?

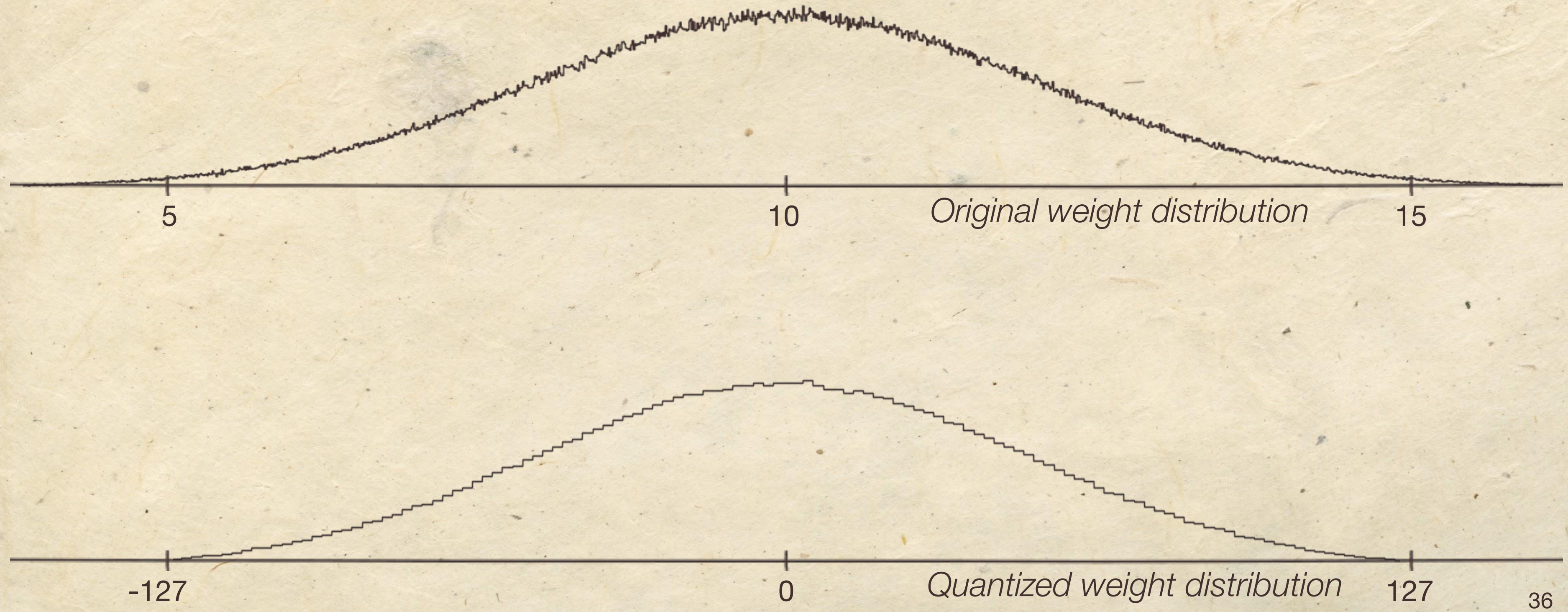


4x smaller
2-4x faster

How to compress Float32 into Int8?

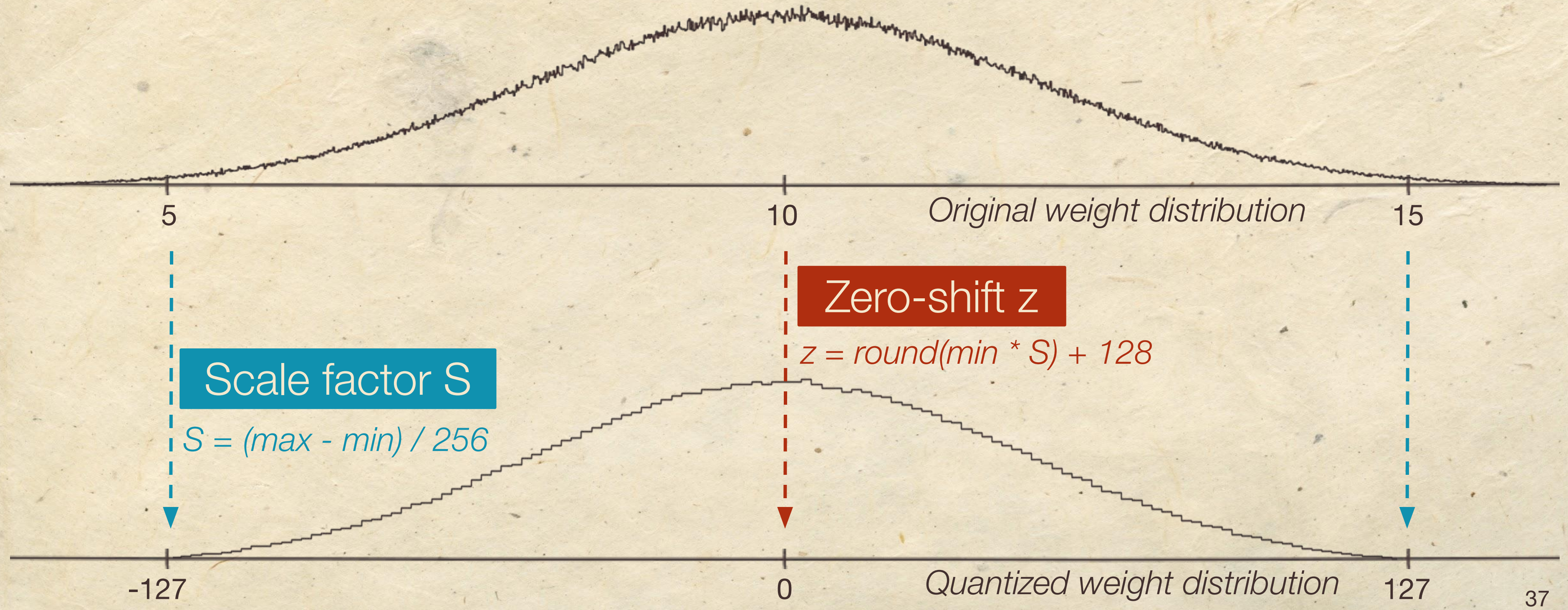


How to compress Float32 into Int8?



How to compress Float32 into Int8?

quantized = S^{-1} * (original + z)



Post-training quantization of weights

For each layer / channel / etc

1. Analyze weight distribution and calculate S and z
2. Apply quantization formula and store quantized weights

→ 4 x smaller weights

What happens during computation?

Post-training quantization of activations

For each layer / channel / etc

1. Run forward pass with a few samples
2. Analyze activation distribution and calculate S and z

→ 2-4 x faster inference

Can we go even smaller with quantization?

- yes, 6-bit, 4-bit or even 2-bit quantization are common
- sacrificing capabilities?
  - hard to predict, models vary in their sensitivity
  - capability loss needs to be evaluated experimentally
  - see the model card for recommended variants

→ up to 16x smaller model files

potentially significant quality loss

📙 Example from our companion repo

- run “Large” Language Model on your local machine with **Ollama**
- get different levels of quantization from 🤗
- test and observe (loss of?) capability

```python
for qtype, model_path in quantized_model_paths.items():
    ollama_model_name = f"{model_name}:{qtype}"
    print(f"Creating Ollama model {ollama_model_name}")
    response = ollama.create(
        model=ollama_model_name,
        modelfile=make_model_file(model_path)
    )
    print(response["status"])
```

[11] ✓ 13.3s Python

```
Creating Ollama model rocket-3B-GGUF:Q8_0
success
Creating Ollama model rocket-3B-GGUF:Q4_K_M
success
Creating Ollama model rocket-3B-GGUF:Q2_K
success
```

The open LLM ecosystem thrives on quantization

- quantization enables
  - medium-sized models on modest hardware (e.g. 15B parameters in 9 GB of RAM)
  - online distribution
- many models in **Ollama** catalog are quantized by default

Quantization at a glance

	Transfer learning	Distillation	Quantization
Model capability	different task	potentially same *	potentially same *
Model size / inference cost	same *	much smaller *	much smaller *
Training data and cost	less *	less *	much less *
Development effort	simple	complex	simple

* compared to original model

Microbudget methods at a glance

	Transfer learning	Distillation	Quantization
Model capability	different task	potentially same *	potentially same *
Model size / inference cost	same *	much smaller *	much smaller *
Training data and cost	less *	less *	much less *
Development effort	simple	complex	simple

https://github.com/ai-dojo/microbudget

* compared to original model

BACKUP

Forward-pass with a quantized model

Our normal float32 forward pass looks like this:

$$y = w \cdot x + b$$

Let's plug in our quantization mapping (^ = quantized):

$$S_y * \hat{y} - z_y = (S_w * \hat{w} - z_y) \cdot (S_x * \hat{x} - z_x)$$

Thanks to the rules of matrix multiplication ·, we get:

$$\hat{y} = z_y + (S_w * S_x / S_y) * ((\hat{w} - z_y) \cdot (\hat{x} - z_x))$$

float32 scalar multiplication   *int8 matrix multiplication*