

Adapting and fine-tuning LLMs

a hands-on approach for business and science

Domen Vreš, Živa Štebljaj

24. 9. 2026 @ UL FRI

Structure of the workshop

9:00 - 12:00 Lectures with practical demonstrations
3 x (45’ + 15’ break)
12:00 - 13:00 Lunch
13:00 - 15:00 Practical work: guided work on a sample task

Content overview

An overview of LLMs



Adaptation and finetuning



Practical work

Workshop materials & resources



Slides



Environment



Code

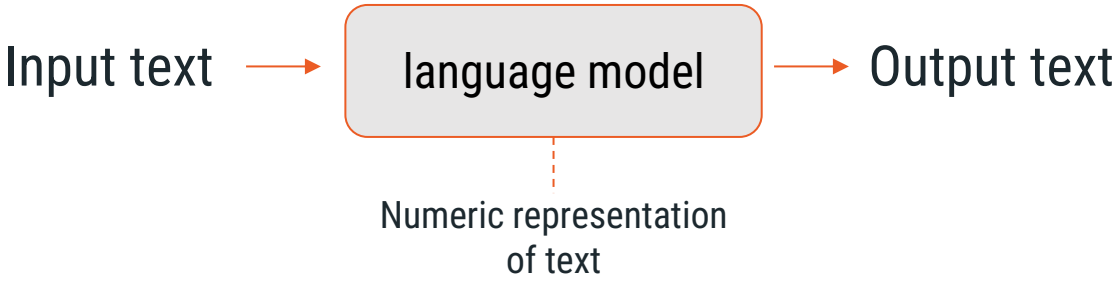
KCUI | Kompetenčni center za umetno inteligenco

Part 1: Language models

A quick revision of the essentials

Language modeling

- language modeling (**abstract**) = modeling of language: understanding and generating natural language



- language modeling (**objective**) = next/missing word prediction

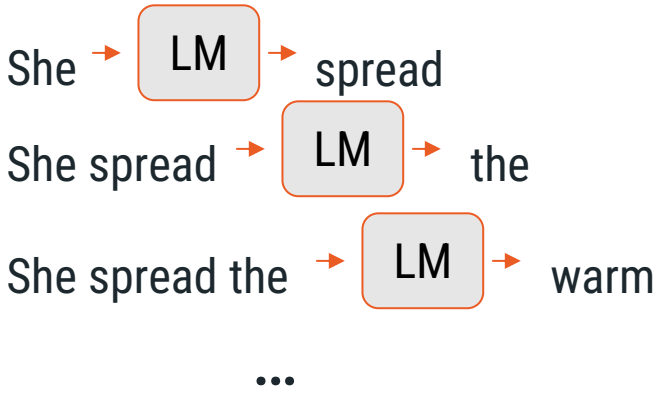
She spread the warm bread with _____ …

She _____ the warm bread with …

many different formulations

- language modeling (objective) = next/missing word prediction
  - trained on large text datasets to **learn language structure and meaning**
  - self-supervised learning

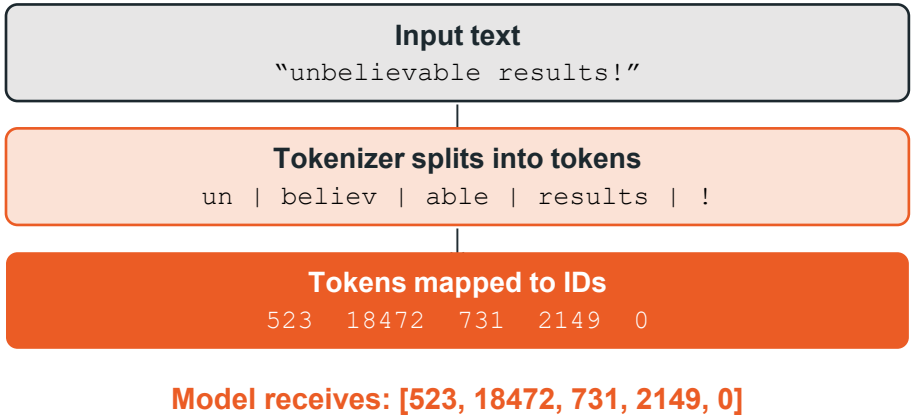
She spread the warm butter with jam.



Self-supervised learning using a large text corpus enables the model to **approximate language understanding through co-occurrence statistics**.

Tokens

Models do not read text directly.
Text is first split into **tokens**, which are mapped to integer IDs from a fixed vocabulary.



Word-based (1 token)
`["unbelievable"]`

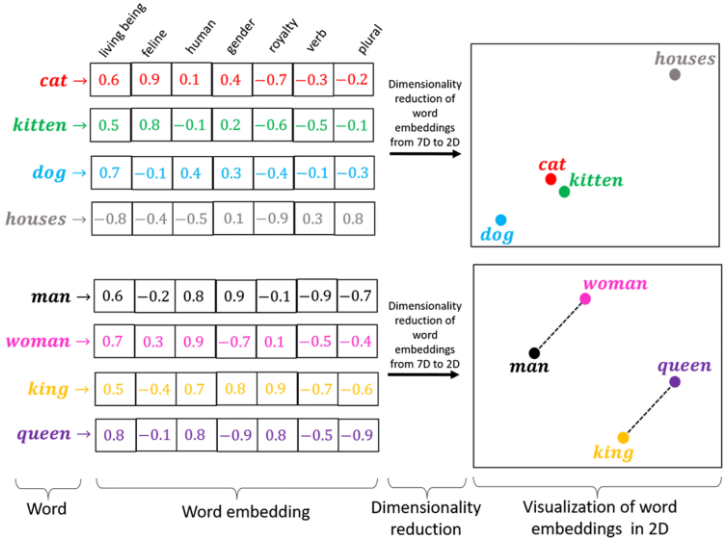
Character-based (12 tokens)
`u-n-b-e-l-i-e-v-a-b-l-e`

Subword (3 tokens) – used here
`["un", "believ", "able"]`

Different models use different tokenizers and vocabularies.

Embeddings

- **word embeddings**: learned dense meaning representations
  - words with similar meaning assigned similar vectors
- **distributional hypothesis**: meaning of the word can be determined based on the context in which it appears in
  - by-product of training language models



From token ID to embedding vector:

Token ID
523

→

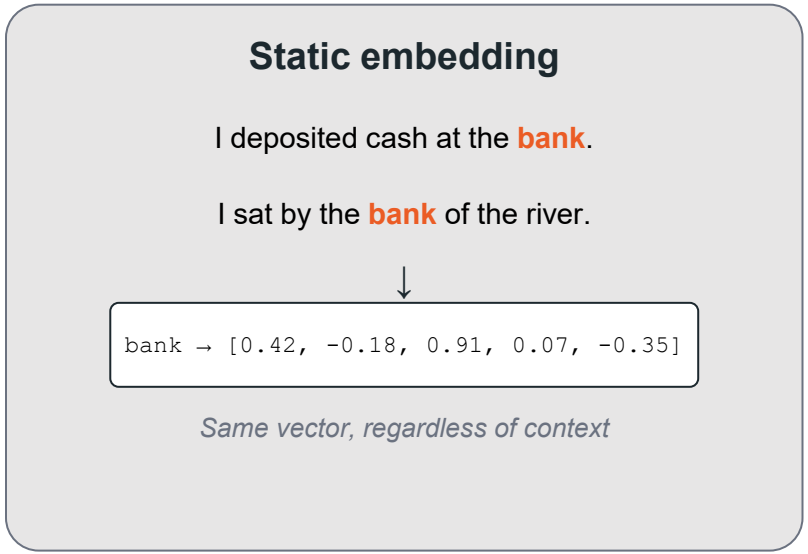
Embedding table lookup
`table[523] → vector`

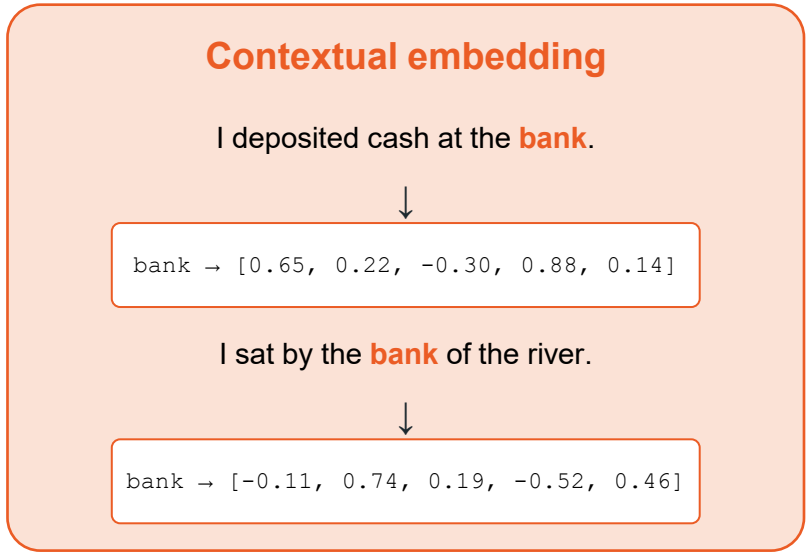
→

Dense vector
`[0.12, -0.87, 0.44, ...]`

Contextual embeddings

- Contextual representations: meaning depends on context

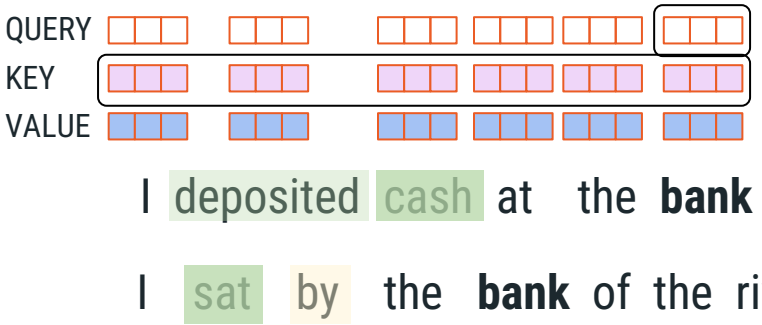




Attention

Transformer-based LMs compute context-aware vectors using self-attention:

- **self-attention mechanism**: selective retrieval of relevant context
  - **Query** = “What information am I looking for?”
  - **Key** = “What kind of information do I contain?”
  - **Value** = “If someone decides I'm relevant, what information should I contribute?”

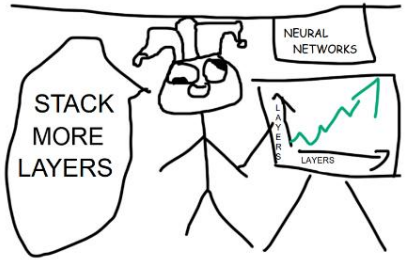


Transformer-based LMs

- **transformer models** = multi-headed attention, multiple layers
  - limited input length (context size): from 512 to 4K to 128K tokens

- different types of models motivated by different task families:
  - **encoder-only** (classification): e.g., BERT models
  - **decoder-only** (generation): e.g., GPT models
  - **encoder-decoder** (seq2seq): e.g., T5 models

- to support this, special tokens are added into the vocabulary such as [CLS] and [MASK]



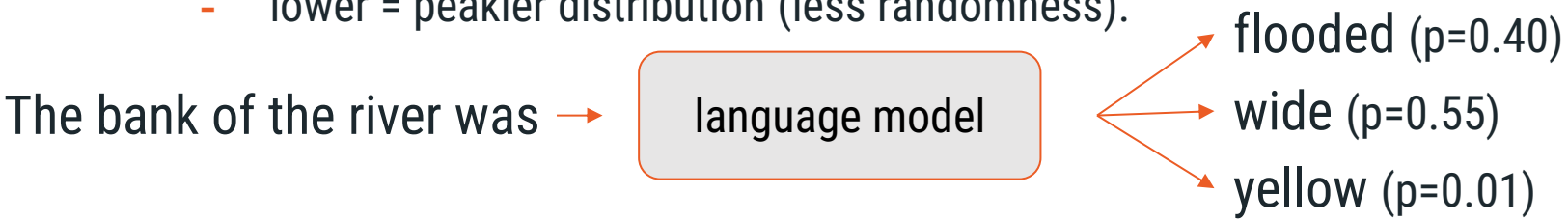
Inference

- language modeling = next word prediction
- many practical tasks can be translated into this setting
  - chatting: P(<answer>|<query>),
    classification: P(<label>|sequence), …
  - **inference: turning probability distributions into outputs**



Inference options

- strategies of selecting the next word:
  - **greedy**: word with highest probability,
  - **top-k**: sample from k most probable words,
  - **top-p**: sample from most likely words whose cumulative probability exceeds p
- **temperature**: adjusts the probability distribution.
  - higher = more uniform distribution (more randomness),
  - lower = peakier distribution (less randomness).



demo

Changing model behaviour

Prompting

What changes: Instructions/context

Best for: Steering the model for a specific request

e.g., rewriting tone/format, one-off extraction, fast prototyping

→

RAG / Tools

What changes: Information available at inference

Best for: External, private, or changing knowledge

e.g., Q&A over internal docs, live data lookups, citing sources

→

Fine-tuning

What changes: Model parameters

Best for: Persistent task-, domain-, or behaviour-specific adaptation

e.g., consistent style, narrow repeated tasks, accuracy beyond prompting's ceiling

Increasing cost, data & engineering effort

When to fine-tune?

Fine-tune when:

- ✓ domain-specific terminology or style
- ✓ high-volume, repeated task with a stable pattern
- ✓ behaviour/knowledge that stays stable over time
- ✓ consistent, structured output

Don’t fine-tune when:

- ✗ one-off or low-volume request
- ✗ needs external, private, or frequently changing information
- ✗ needs quick iteration without retraining

Examples

“Decline a vendor’s price increase, politely, just this once.”

“Extract invoice fields, matching 3 example formats.”

“Answer questions on this week’s updated expense policy.”

“Fetch live sensor readings before answering.”

“Sort 5,000 daily tickets into 15 fixed categories.”

“Extract compound, concentration & conditions from papers.”

“Draft on-brand support replies, at scale.”



Efficient LLM adaptation

Fewer parameters to update

- **PEFT:** freeze the base model, train a small add-on

Reduce memory per step

- **Mixed precision** (BF16/FP16): lower-precision weights & activations
- **Quantization:** compress weights to fewer bits
- **Gradient checkpointing:** recompute activations instead of storing them

Scale across hardware & steps

- **Batching** (micro vs. global): samples per step vs. per update
- **Gradient accumulation:** simulate a larger batch, same memory
- **Parallelism:** split work across GPUs

KCUI | Kompetenčni center za umetno inteligenco

Part 1.2: Data

The importance of (good) data

- in machine learning, data quality makes or breaks the model: "garbage in, garbage out"
- to build a successful model, we need **data representative of the problem we are trying to solve**
  - what is good data? what is bad data?
  - **how much high quality data do we want?**



10 clean texts
0 offensive texts



10 clean texts
10 offensive texts



1000 clean texts
1000 offensive texts

What if these are all from one niche subreddit?

Quality of data is partially verified through experimentation.

The importance of (good) data, part 2

- in addition to training data, we also need (unseen) **evaluation data**
- typically three splits: training, validation, test data
  - training data: used to train the model
  - validation data: used to fine-tune hyperparameters of the model
  - test data: used to observe model success “in the wild”

- usually majority of data used for training, often ~10-20% of data used for validation and testing, each

Data quality issues

- **Duplicates:** repeated examples inflate certain patterns
- **Inconsistent formatting:** inputs/outputs don’t follow the same structure
- **Train/test leakage:** the same or near-identical examples appear in both splits
- **Label noise:** incorrect or inconsistently applied labels
- **Class imbalance:** some categories are heavily over- or under-represented

Dataset sources

Existing datasets

CLARIN.SI, Hugging Face Hub, Kaggle, shared tasks. Check the licence, domain and how the labels were made.

Annotate your own

Fits your task exactly, but slow and costly. Needs guidelines, several annotators and an agreement check.

Synthetic data

Written or labelled by an LLM. Cheap and fast, but only as good as the model that produced it.

Synthetic data in low-resource settings (e.g. Slovene)

Common approaches

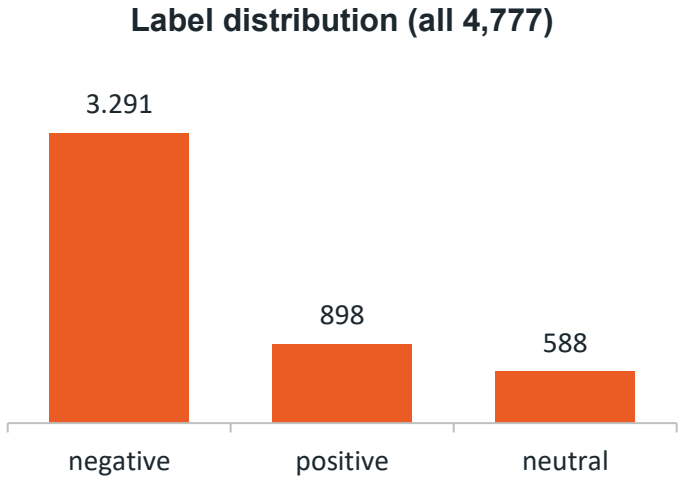
- **Translate** English datasets with machine translation
- **Generate** examples or instructions with a strong LLM
- **Label** real unlabelled text with an LLM (“silver” labels), then check a sample by hand

Watch out for

- **Unnatural language:** translationese, repetitive and less diverse text
- **Inherited errors:** the generator’s mistakes and biases, often worse in low-resource languages
- **Licences:** terms of use of the generator model and source data
- **Evaluation:** always test on real, human-labelled data

Our example: KKS 1.001

- **Opinion corpus of Slovene web comments** (CLARIN.SI)
- **4,777 comments** from 4 news portals (rtvslo, 24ur, finance, reporter), 2007–2015
- **3 labels:** negative / neutral / positive
- **Split:** stratified random 80/10/10 (seed 42) → 3,821 / 478 / 478
- **Our notebook keeps only** text + label. It drops Source, Category, User, article URL, date, codersNum and contextRequired.



Class imbalance: a model that always answers “negative” already gets 68.9 % accuracy. Neutral, the rarest class, is only 12.3 %.

KCUI | Kompetenčni center za umetno inteligenco

Part 1.3: Evaluation

Evaluation metrics: quantifying success

- what a successful model is depends on the problem
- **evaluation metric**: approximate measure of success in solving the problem
- e.g., accuracy: proportion of correct predictions out of all predictions

650 correct predictions out of 1000 = 65% accuracy

- **Interpretation**: the importance of selecting the appropriate metric!

  90% of all texts are clean, our model achieves accuracy 65%:
  is our model useless? (F1 score is more suitable for imbalanced data)

- Evaluation metrics can get much more complex (e.g., evaluation of text answers)

Evaluation depends on the task

Task	Example	How to evaluate
Classification	sentiment, topic	accuracy, precision / recall, macro-F1, confusion matrix
Extraction	named entities, invoice fields	exact or partial match, F1 per field or span
Generation with a reference	translation, summarisation	overlap (BLEU, chrF, ROUGE) or learned metrics (COMET, BERTScore)
Open-ended generation	chat, support replies	human evaluation, LLM-as-a-judge, benchmarks with fixed answers

LLM-as-a-judge: a strong LLM grades the outputs of another model

- **Pointwise:** score one answer against a rubric, e.g. 1–5 for correctness and fluency
- **Pairwise:** pick the better of two answers (A vs. B)
- **Pros:** cheap, fast, scales to thousands of outputs
- **Known biases:** prefers the first answer, longer answers and its own model family
- **Mitigate:** clear rubric, swap answer order, check against human ratings on a sample
- **Low-resource languages:** the judge may be weaker than in English

Evaluating our sentiment classifier

Model (test split, 478 comments)	Accuracy	Macro-F1
Always “negative” (majority class)	68.8 %	0.27
Random guess with class frequencies	≈ 52 %	≈ 0.33
Your fine-tuned CroSloEngual BERT	**?**	**?**

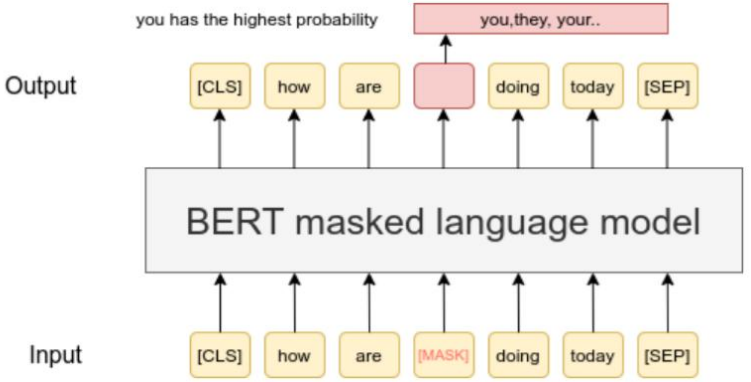
Why macro-F1 here? The majority baseline scores 68.8 % accuracy but only 0.27 macro-F1. Macro-F1 averages the three classes equally, so ignoring neutral and positive costs you. Beat both baselines before you celebrate.

KCUI | Kompetenčni center za umetno inteligenco

Part 2.1:
Fine-tuning BERT models

Pre-training Models

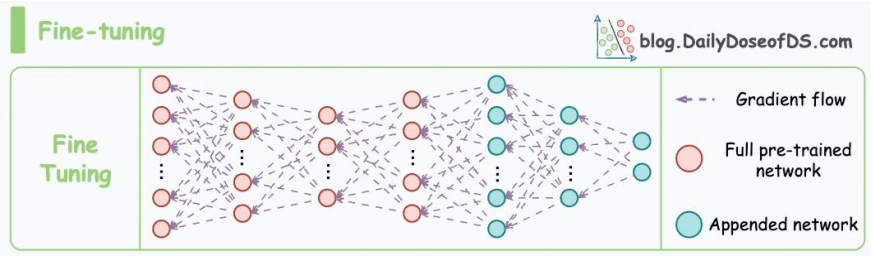
- **Self-supervised learning** involves learning from unlabeled data, using the data itself to create training objectives.
- **Next Token Prediction**: The model predicts the next word in a sequence based on the previous words.
- **Masked Language Modeling (MLM):** Certain words in a sentence are masked or hidden, and the model learns to predict these masked words using the surrounding context.
- **Result**: A general knowledge/understanding of language.



Source: https://raw.githubusercontent.com/UKPLab/sentence-transformers/master/docs/img/MLM.png

Fine-tuning models

- **Fine-tuning**: Adapting a pre-trained model to specific tasks by further training on a smaller, task-specific dataset.
- On top of LM, we attach e.g. classification head.
- Key Steps:
  1. **Start with a Pre-trained Model**: Utilize a general-purpose model trained on large datasets.
  2. **Prepare Task-Specific Data**: Collect relevant, labeled data for the specific task.
  3. **Fine-Tune the Model**: Train with a low learning rate to adjust model weights for the task.



Learning rate scheduler: cosine decay with linear warmup

Why schedule the learning rate?

- Too high LR early → unstable loss, divergence
- Too high LR late → overshooting minima
- Gradual ramp-up + cosine decay solves both

Phase 1: Linear warmup

LR rises linearly: 0 → max_lr over Warmup steps
Typical: 10 % of total steps

Phase 2: Cosine decay

LR follows cosine curve: max_lr → min_lr
min_lr is typically 10% of max_lr

Cosine LR with linear warmup



Batch size: per-device vs. global

Per-device (micro) batch size

- Samples processed on ONE GPU per forward pass
- Limited by GPU VRAM (activation memory grows linearly)
- Typical values: 1–8 for LLM fine-tuning

Global batch size

- Samples seen before one weight update
- Larger global batch → stabler gradient estimates
- Too large → sharp minima, poorer generalization
- Typical values: 32, 64, 128

Example: 4 GPUs, micro-batch = 4, no accumulation



→ **global batch = 16 samples per update**

Gradient accumulation

Why?

- Memory limits how big a micro-batch can be
- Gradient accumulation simulates a larger batch without increasing memory usage
- Gradients are summed across mini-steps

G = 4 accumulation steps, micro-batch = 2

Step 1	Step 2	Step 3	Step 4
forward + accum grad	forward + accum grad	forward + accum grad	forward + accum grad

$\nabla L = \nabla L_1 + \nabla L_2 + \nabla L_3 + \nabla L_4$ (accumulated gradients)

`optimizer.step()` → weights updated once

Effective batch = 4 steps × 2 samples = 8 per GPU

Global batch size formula

$$\text{global_batch} = \text{micro_batch} \times \text{N_GPUs} \times \text{accum_steps}$$

KCUI | Kompetenčni center za umetno inteligenco

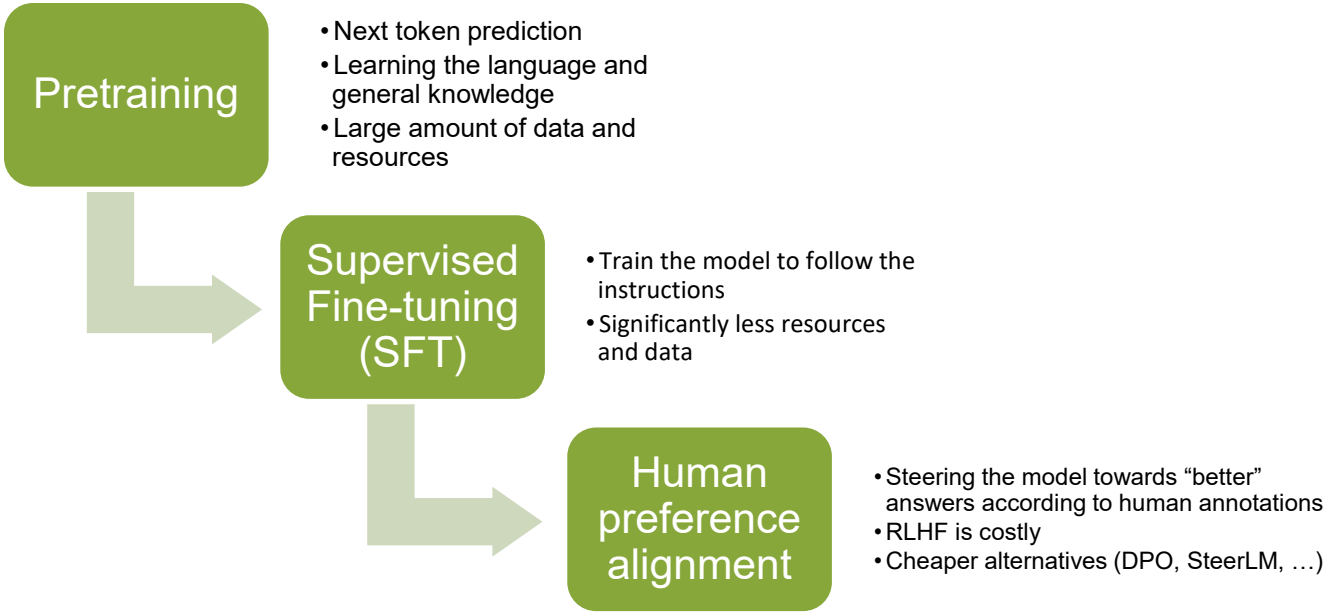
Practical example

KCUI | Kompetenčni center za umetno inteligenco

Part 2.2: Fine-tune generative models

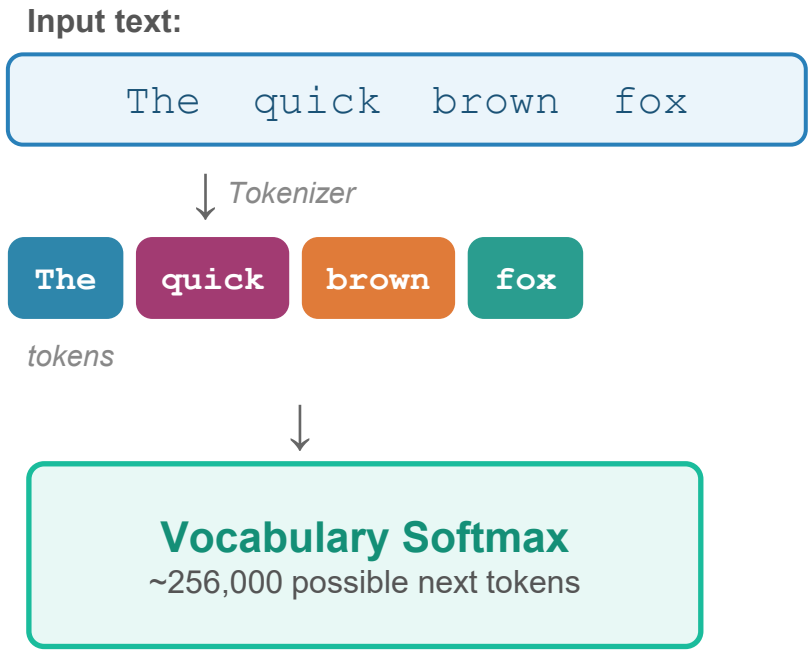
- Theory
- Techniques
- Practical advice

From next token prediction to a chat model



Next token prediction

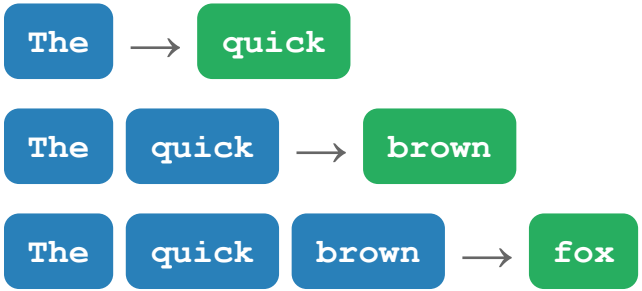
- Text generators are classifiers
  - At each step: predict the most probable next token
- Output layer = vocabulary size (~256K tokens)
  - Some models share embedding and output weights
- Trained on massive text corpora (pre-training)



Autoregressive training

Context (seen tokens)

Loss contribution



… *continues for every position in training data*

Training objective

$$\mathcal{L}(\theta) = \mathrm{E}[-\sum_i \log p(x_i \mid x_0, \ldots, x_{i-1}\,; \theta)]$$

Minimize avg. negative log-likelihood of the correct next token across all positions

Supervised fine-tuning

From prediction to instruction-following

- Adapt next-token prediction to follow instructions
- Training data: (prompt, completion) pairs
- Prompt = instruction + optional context
- Loss computed only on completion tokens
- Prompts and completions separated by special strings or tokens
- Chat template

Alpaca-style training data format

PROMPT

```
Below is an instruction.
Write an appropriate response.

### Instruction:
Translate "Hello, world!" to French.

### Response:
```

↓

COMPLETION

```
Bonjour, le monde !
```



Philippe Schrettenbrunner, based on the xkcd comic "Exploits of a Mom (327)"



Chat template (Gemma 2 example)

```
<bos>
<start_of_turn>user
Explain what a large language model is.
<end_of_turn>
<start_of_turn>model
A large language model is a neural network trained on large amounts of text.
<end_of_turn>
<start_of_turn>user
Give me a simple analogy.
<end_of_turn>
<start_of_turn>model
Think of it like a very well-read autocomplete that learned patterns from many books.
<end_of_turn>
```

SFT training

PROMPT TOKENS

<s> | Below is | ###Inst: | Translate | Hello... | ###Resp:

no gradient (prompt masked from loss)

COMPLETION TOKENS

Bonjour, | le monde | !

loss computed ✓

The model processes the full sequence but gradients flow only through completion tokens.

SFT Training Objective

$$\mathcal{L}(\theta) = -\sum_i \log p(y_i \mid y_0, \ldots, y_{i-1}, x; \theta)$$

y = completion tokens · x = prompt (masked) · θ = model parameters

Preference alignment

Teaching models to produce 'better' responses

- "Better" is defined by human preference annotations
- Dataset: (prompt, chosen response, rejected response) triples
- Dimensions: helpfulness, correctness, safety, toxicity, …
- Iterative — model can be refined across multiple rounds

Collecting preference data

PROMPT: "What is a neural network?"

↓ *Model samples multiple responses from its output distribution*

Response A
A computational model that learns patterns in data using layers of connected nodes.

Response B ✓
A system inspired by neurons that learns hierarchical patterns, enabling tasks like recognition and generation.

Response C ✗
It's like a computer brain thing. Scientists use it for AI stuff.

Human annotator ranks: B > A > C → training pair: (B chosen, C rejected)

Preference alignment methods

Comparative

Compare two or more outputs

- **PPO** – Proximal Policy Optimization
  Needs a reward model (RLHF)
- **DPO** – Direct Preference Optimization
  Simpler, no reward model needed
- **GRPO** – Group Relative Policy Opt.
  Used in DeepSeek-R1

Descriptive

Score each output independently

- **SteerLM**
  Annotate responses with quality attributes (helpfulness, safety, etc.)

DPO: Direct Preference Optimization

✓ Chosen response
Higher quality / more helpful output

✗ Rejected response
Lower quality / less preferred output

DPO directly optimizes the policy without a separate reward model — increasing the relative probability of chosen over rejected responses.

DPO Training Objective

$$\mathcal{L}(\theta) = -\mathrm{E}(x,y^{+},y^{-}) \left[\log \sigma(\beta \cdot \Delta(x, y^{+}, y^{-}; \theta)) \right]$$

$$\text{where } \Delta = \log \pi_\theta(y^{+}|x)/\pi_\mathrm{ref}(y^{+}|x) - \log \pi_\theta(y^{-}|x)/\pi_\mathrm{ref}(y^{-}|x)$$

y^{+} = chosen · y^{-} = rejected · β = regularization · π_ref = frozen reference model

DPO: training example

PROMPT

What are the risks of not getting enough sleep?

✓ CHOSEN RESPONSE

Sleep deprivation has serious health consequences.

Short-term: impaired cognition, reduced reaction time, mood disturbances, difficulty concentrating.

Long-term: increased risk of cardiovascular disease, diabetes, obesity, and weakened immune function.

The CDC recommends adults get 7+ hours per night.

✗ REJECTED RESPONSE

Not sleeping enough can make you tired.

Some people can function on less sleep, so it depends on the person. If you feel fine then it is probably okay.

Just drink coffee to stay awake if you need to get things done.

Training memory: what has to fit in GPU RAM?

Parameter storage (BF16 training, Adam optimizer)

Model weights	2 bytes × N params	**2N**
Gradients	2 bytes × N params	**2N**
Optimizer (Adam)	12 bytes × N params (fp32)	**12N**

Total (parameters) ≈ 16 × N

\+ *activations (scale with batch size and context length)*

Attention: quadratic memory scaling

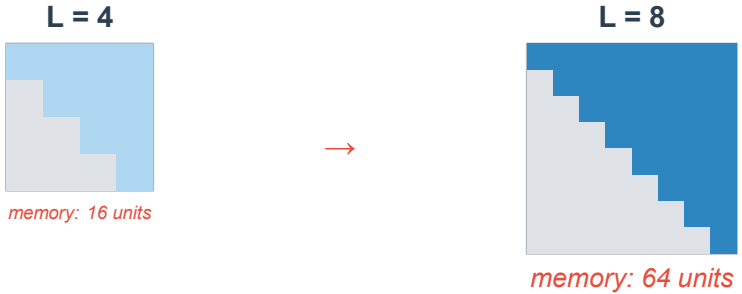
Standard self-attention computes an L × L score matrix:

Attention memory

Attention matrix: $L^2 \times H \times 4$ bytes

L = context length, H = number of heads

Doubling context length → 4× more attention memory:



Memory in practice: GaMS-9B (8k context, BF16)

$N = 9 \times 10^9$ parameters · BF16 precision · micro-batch size = 1 · context = 8 192 tokens

Component	Calculation	Memory
Model weights	2 bytes × 9B	18 GB
Gradients	2 bytes × 9B	18 GB
Adam optimizer	12 bytes × 9B	108 GB
Activations	≈ layers × heads × d × L^2	~200 GB
Total (model + grads + optimizer + activations):		**≈ 350 GB**

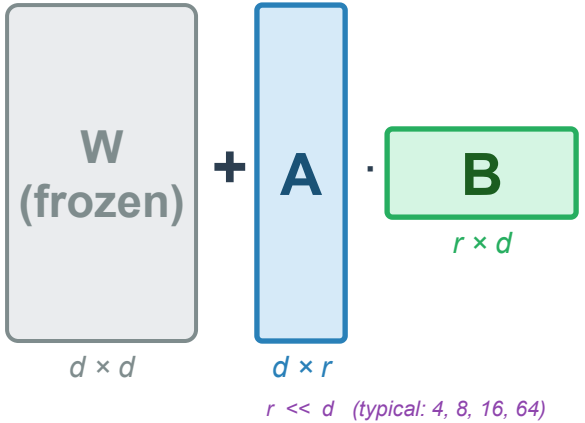
→ **We need parameter-efficient methods like LoRA!**

LoRA: Low-Rank Adaptation

Core idea

- Base model weights W are frozen
- Learn a small additive update: W + ΔW
- ΔW is decomposed into two small matrices:
  - ΔW = A · B (rank-r decomposition)
- Only A and B are trained

Weight update decomposition



Parameter reduction example (d = 4 096, r = 8)

Full weight matrix W: d × d = 4 096 × 4 096

LoRA matrices A, B: d × r + r × d = 2 × 4 096 × 8

Reduction: 256× fewer trainable parameters!

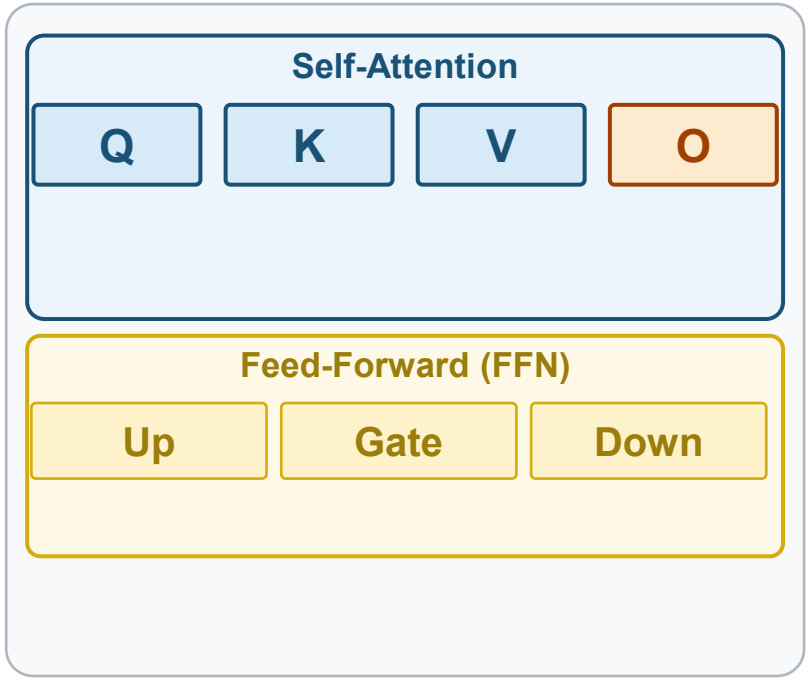
Forward pass

$$\text{output} = (W + \alpha/r \cdot A{\cdot}B) \cdot \text{input}$$

α = scaling factor (alpha)

LoRA: where it's applied

Transformer block (decoder layer)



Key hyperparameters

rank r
Dimensionality of LoRA matrices. Common: 4, 8, 16, 64. Higher r → more params, more capacity.

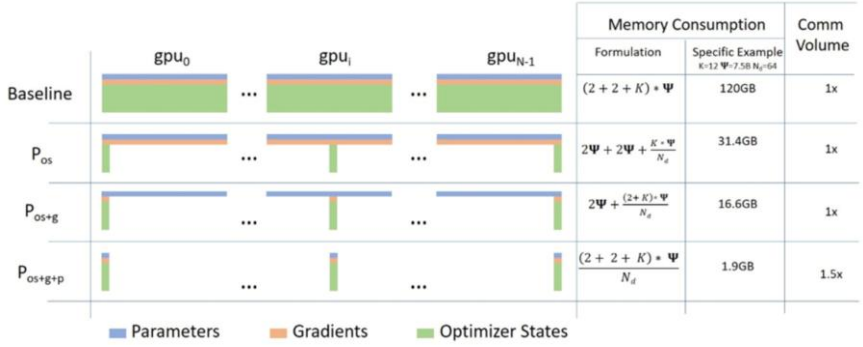
alpha α
Scaling factor. Effective scale = α / r. Often set α = r or α = 2r.

target_modules
Which weight matrices get LoRA. Typically q_proj, v_proj; sometimes all attention + FFN.

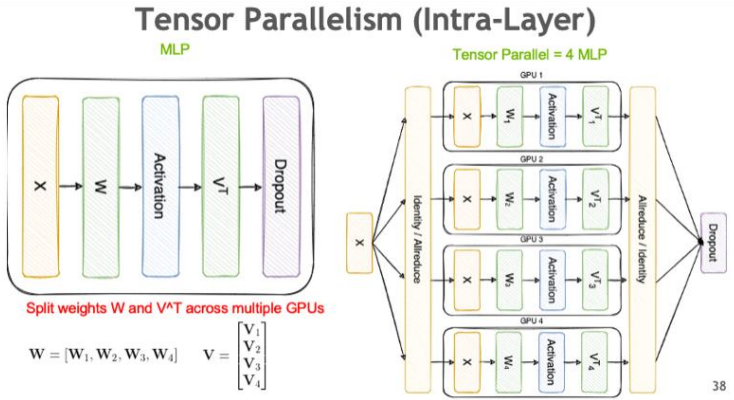
lora_dropout
Dropout applied to LoRA path. Small value (0.05–0.1) helps regularization.

Multi GPU (or node) training

- Official SLAIF service: https://edu.slaif.si/workflows/GaMS-PEFT-Tuning
  - The whole setup for Vega cluster (we can discuss if you have your own infrastructure)
- Various parallelisms
  - DeepSpeed ZeRO stages
  - Megatron-Bridge (NeMo)



	Memory Consumption		Comm Volume
	Formulation	Specific Example $K=12\ \Psi=7.5B\ N_d=64$	
Baseline	$(2+2+K)*\Psi$	120GB	1x
P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x



KCUI | Kompetenčni center za umetno inteligenco

Practical example

Hvala za udeležbo!

Thank you for attending!

KCUI | Kompetenčni center za umetno inteligenco

IZUM
Institute of Information Science, Maribor, Slovenia

arnes

University of Maribor

Gospodarska zbornica Slovenije
Chamber of Commerce and Industry of Slovenia

Jožef Stefan Institute
Ljubljana, Slovenia

UNIVERZA V LJUBLJANI
University of Ljubljana

FIS Fakulteta za informacijske študije
Faculty of information studies

TP LJ01

Financerja / Financed by:

REPUBLIKA SLOVENIJA
MINISTRSTVO ZA VISOKO ŠOLSTVO, ZNANOST IN INOVACIJE

EuroHPC
Joint Undertaking

Projekt SLAIF: Slovenska tovarna umetne inteligence je finančno podprlo Ministrstvo za visoko šolstvo, znanost in inovacije. Projekt je bil na razpisu skupnega podjetja EuroHPC izbran za financiranje v okviru programov Obzorje Evropa ter Digitalna Evropa.

SLAIF: Slovenian AI Factory has been funded by the Ministry of Higher Education, Science and Innovation of Republic of Slovenia. At a call by EuroHPC JU, the project has received a positive funding decision under Horizon Europe and Digital Europe Programmes.