LLM Pretraining

Tópicos em Ciência de Dados

Prof. Dr. Denis Mayr Lima Martins

Pontifícia Universidade Católica de Campinas

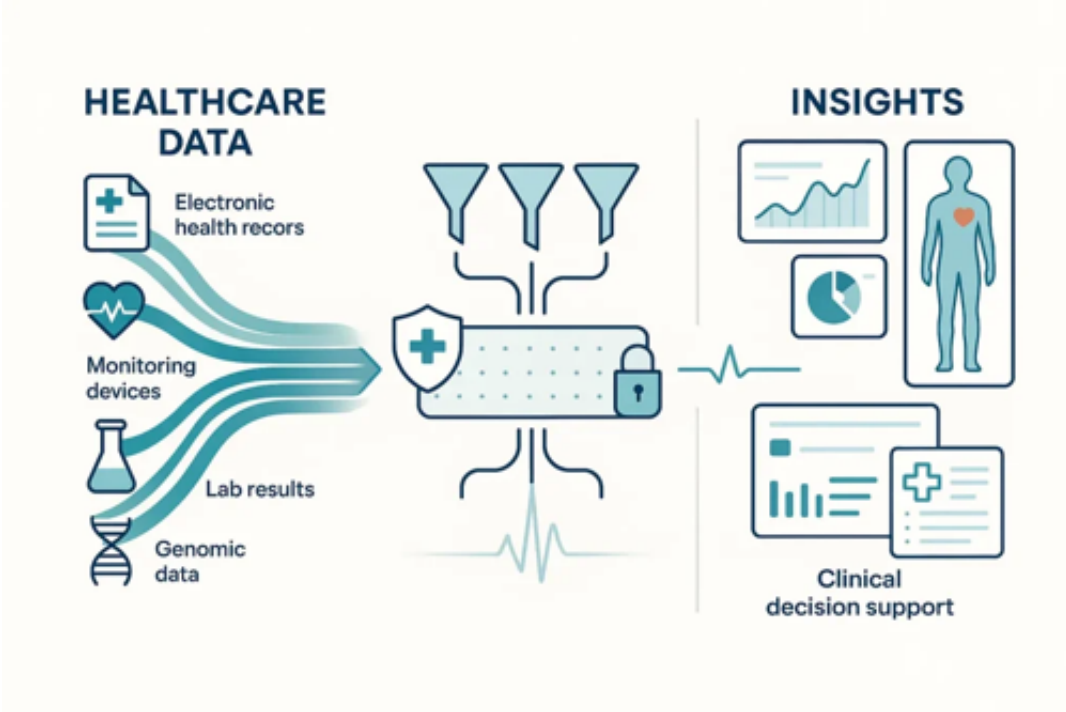
Objetivos de Aprendizagem

- Compreender o conceito de Foundation Models e suas aplicações.
- Entender a importância de pré-treinamento na criação de Large Language Models (LLMs)
- Compreender os princípios e práticas de pré-treinamento de LLMs.
- Implementar pré-treinamento de um modelo GPT-2.

Baseado no Livro Build a Large Language Model From Scratch de Sebastian Raschka

Code repository: https://github.com/rasbt/LLMs-from-scratch

Qual o **maior** problema em depender **apenas** de aprendizado **supervisionado** tradicional?



Dados anotados (labeled data) na área da Saúde são caros e difíceis de obter em larga escala. Fonte: Macgence.

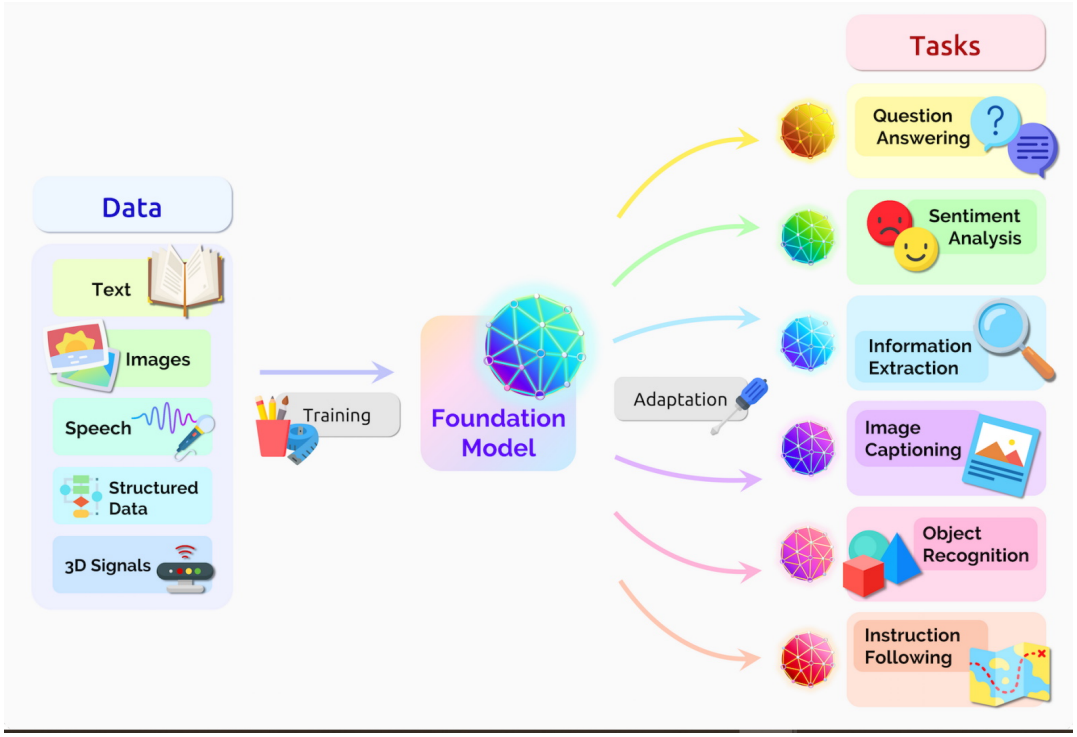
Treinamento Supervisionado Tradicional

- **Escassez de Dados Rotulados:** Obter dados rotulados é caro e demorado.
- **Generalização Limitada:** Modelos treinados em datasets específicos podem ter dificuldades para generalizar para novos cenários.
- **Necessidade de Engenharia de Features:** Requer conhecimento especializado para selecionar e criar features relevantes.

Foundation Models

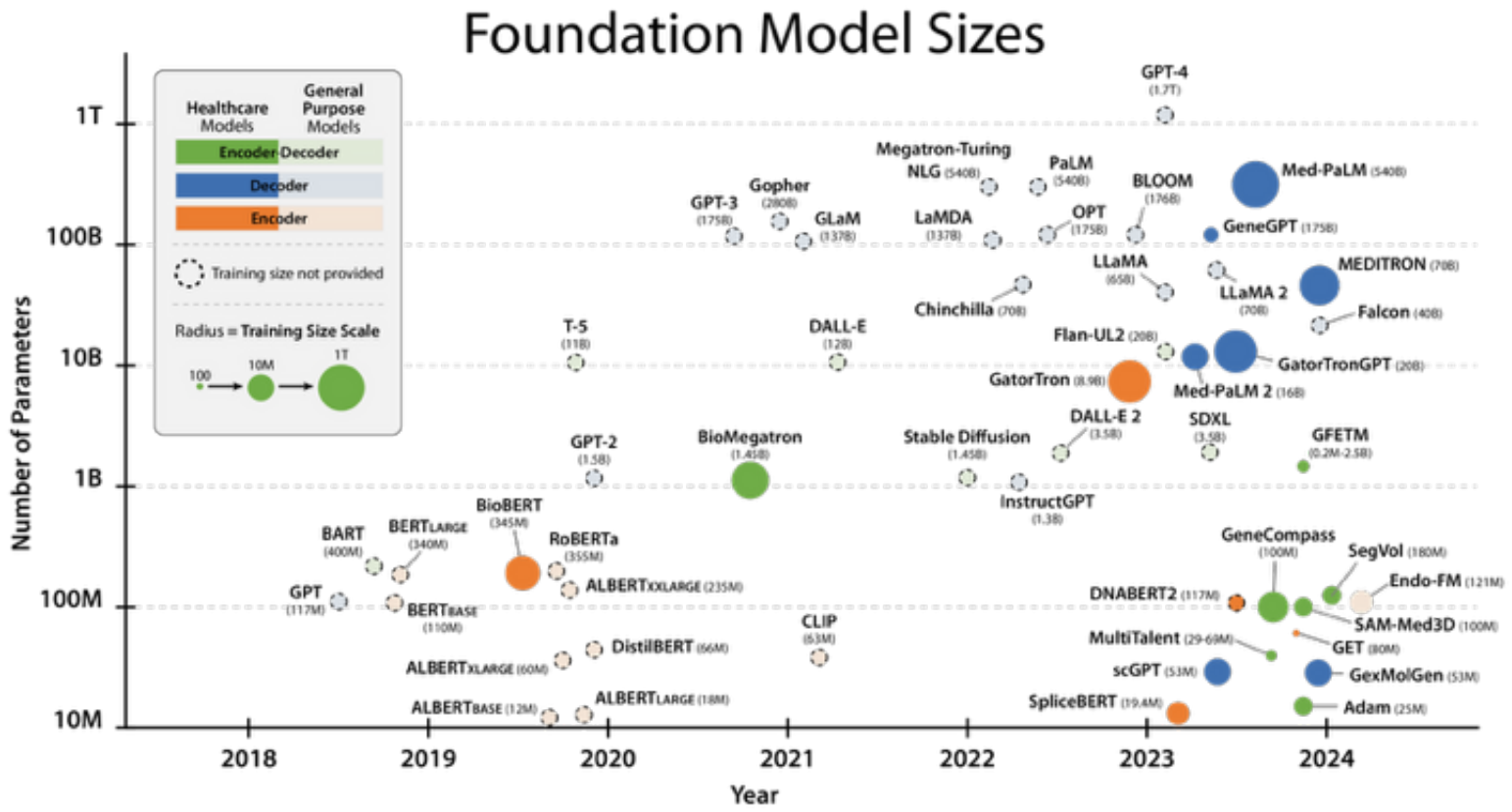
- **Conceito:** Modelos treinados em grandes volumes de dados *não rotulados* que podem ser adaptados para uma variedade de tarefas downstream.
- **Características:**
    - **Escalabilidade:** Treinados em datasets massivos.
    - **Representação**: Aprendem a representar dados capturando nuances intrínsecas que ajudam a generalizar para outras tarefas.
    - **Adaptabilidade:** Podem ser finamente ajustados (fine-tuned) para tarefas específicas.
    - **Emergência de Capacidades:** Demonstram habilidades que não foram explicitamente programadas.
- **Analogia:** Aprender a ler e escrever antes de se especializar em um gênero literário específico.

Foundation Models



Foundation Model. Fonte: On the Opportunities and Risks of Foundation Models.

Foundation Models na Medicina



Learning architecture, model size, and training data used by representative foundation models. Fonte: A Comprehensive Survey of Foundation Models in Medicine.

O quão fácil é utilizar um modelo pré-treinado?

O quão fácil é utilizar um modelo pré-treinado?

Mais de 2 milhões de modelos pré-treinados no https://huggingface.co/models

In [ ]:

```
from transformers import pipeline, set_seed

generator = pipeline(
    'text-generation',
    model='gpt2',
    truncation=True)

set_seed(42)

generator(
    "Hello, I'm a language model,",
    max_length=5,
    num_return_sequences=1,
    pad_token_id=generator.tokenizer.eos_token_id)
```

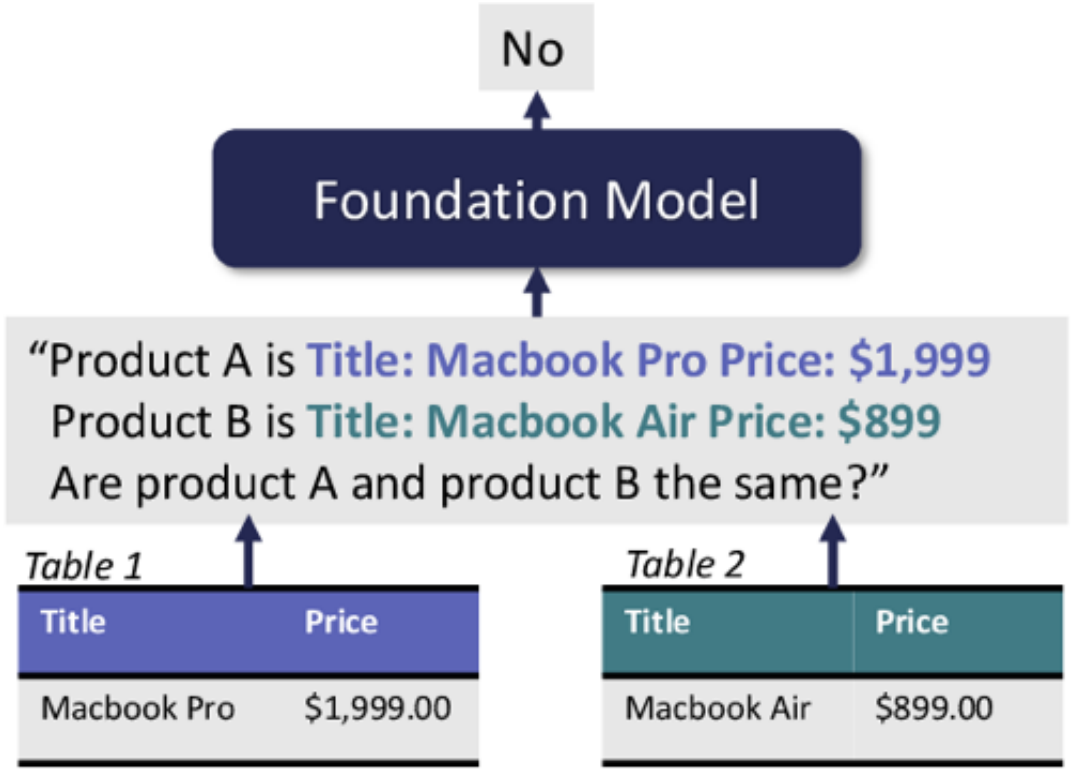
Foundation Models: Objetivos

- **Masked Language Modeling (MLM):** (Ex: BERT)
    - Ocultar aleatoriamente algumas palavras no texto e treinar o modelo para prever as palavras ocultas.
    - **Loss:** Cross-Entropy sobre posições mascaradas.
- **Causal Language Modeling (CLM):** (Ex: GPT)
    - Treinar o modelo para prever a próxima palavra em uma sequência.
    - **Loss:** Cross-Entropy sobre todas as posições, com máscara causal.

Foundation Models: Tipos

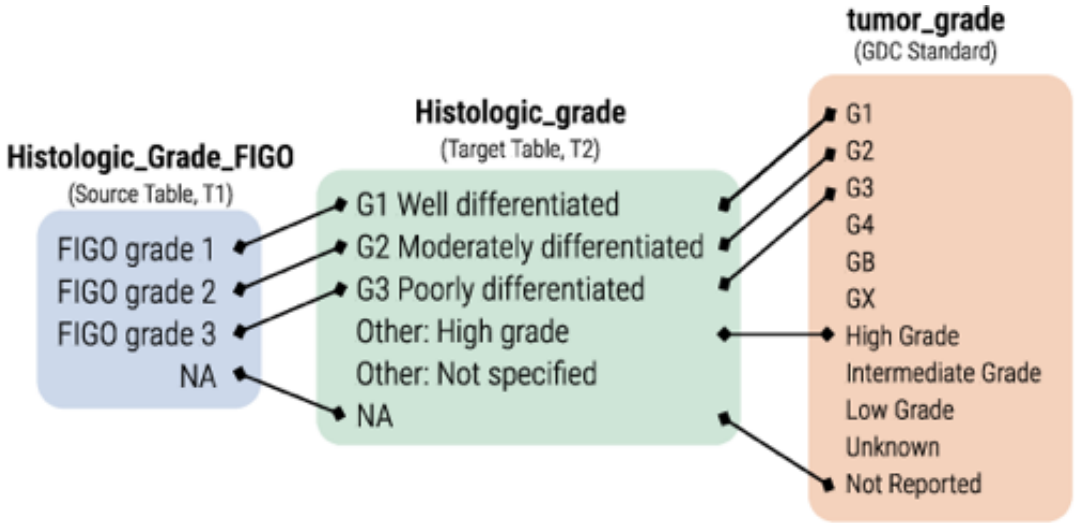
- **Embedding Extractiors**: Transformam dados de entrada em uma representação mais apropriada que preserva as características mais relevantes.
- **Zero-Shot Models**: Executam a tarefa diretamente sobre dados não vistos durante o (pré-)treinamento. Operam sob o princípio de que, mesmo que um modelo não tenha visto um objeto específico durante o treinamento, ele deve conseguir usar seu conhecimento sobre outros objetos semelhantes para identificar o novo.
- Exemplos: Meta Segment Anything Model (SAM), OpenAI CLIP, Meta DINOv2.

Foundation Models e Especialistas em Dados (cont.)



Transformando dados com Foundation Models. Fonte: Can Foundation Models Wrangle Your Data?

Foundation Models e Especialistas em Dados



Integração/Harmonização de Dados via Agentes de LLM. Fonte: Interactive Data Harmonization with LLM Agents: Opportunities and Challenges.

Datasets de Pré-treinamento

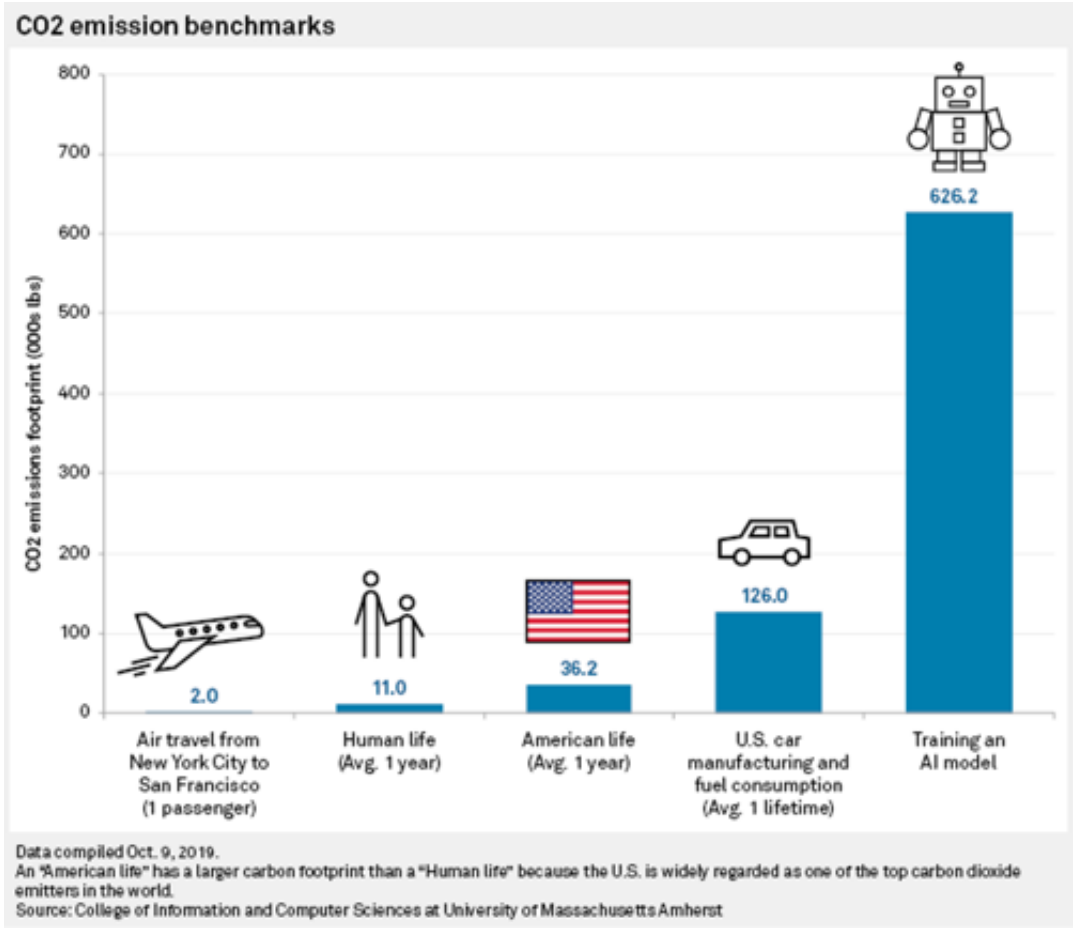
- **Common Crawl:** Um vasto repositório de páginas da web.
- **Wikipedia:** Uma enciclopédia colaborativa online.
- **BooksCorpus:** Um conjunto de livros digitais.
- **The Pile:** Uma coleção diversificada de datasets textuais.
- Project Gutenberg, arXiv, PubMed
- Código (GitHub, GitLab)

Limpeza de dados

- Remover duplicatas, HTML tags, scripts.
- Filtrar linguagem e conteúdo ofensivo.
- Normalizar pontuação, acentos.
- Balanceamento de domínio (ex.: código > texto natural).
- Equalização de gêneros, regiões geográficas.

Desafios e Considerações Éticas

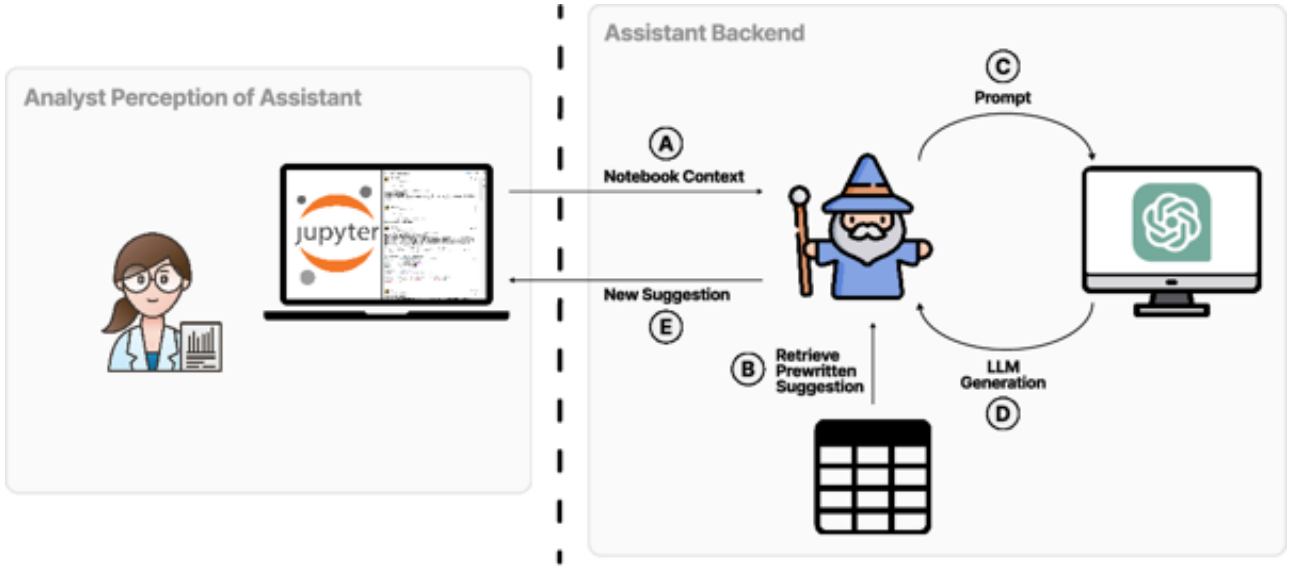
- **Viés nos Dados:** Os modelos podem herdar vieses/preconceitos presentes nos dados de treinamento.
- **Desinformação:** Capacidade de gerar texto convincente pode ser mal-uso.
- **Privacidade:** Modelos podem memorizar trechos sensíveis.
- **Custo Computacional:** O pré-treinamento e o fine-tuning de LLMs exigem recursos computacionais significativos.
- **Impacto Ambiental:** Consumo energético e emissões de CO_2 no treinamento.



Impacto Ambiental da AI. Fonte: Forbes.

Resumo

- LLMs são Foundation Models específicos para geração de texto.
- Pré-treinamento exige grande escala de parâmetros, dados e GPU.
- Ética não pode ser tratada como complemento; é parte integrante do design.
- Impacto Ambiental: https://hbr.org/2024/07/the-uneven-distribution-of-ais-environmental-impacts
- Leituras recomendadas:
    - A Dataset-Centric Survey of LLM-Agents for Data Science
    - How Do Data Analysts Respond to AI Assistance? A Wizard-of-Oz Study
    - The Uneven Distribution of AI's Environmental Impacts
- Pergunta: Qual aspecto do pré-treinamento você considera mais crítico para o desempenho de um LLM? Por que?



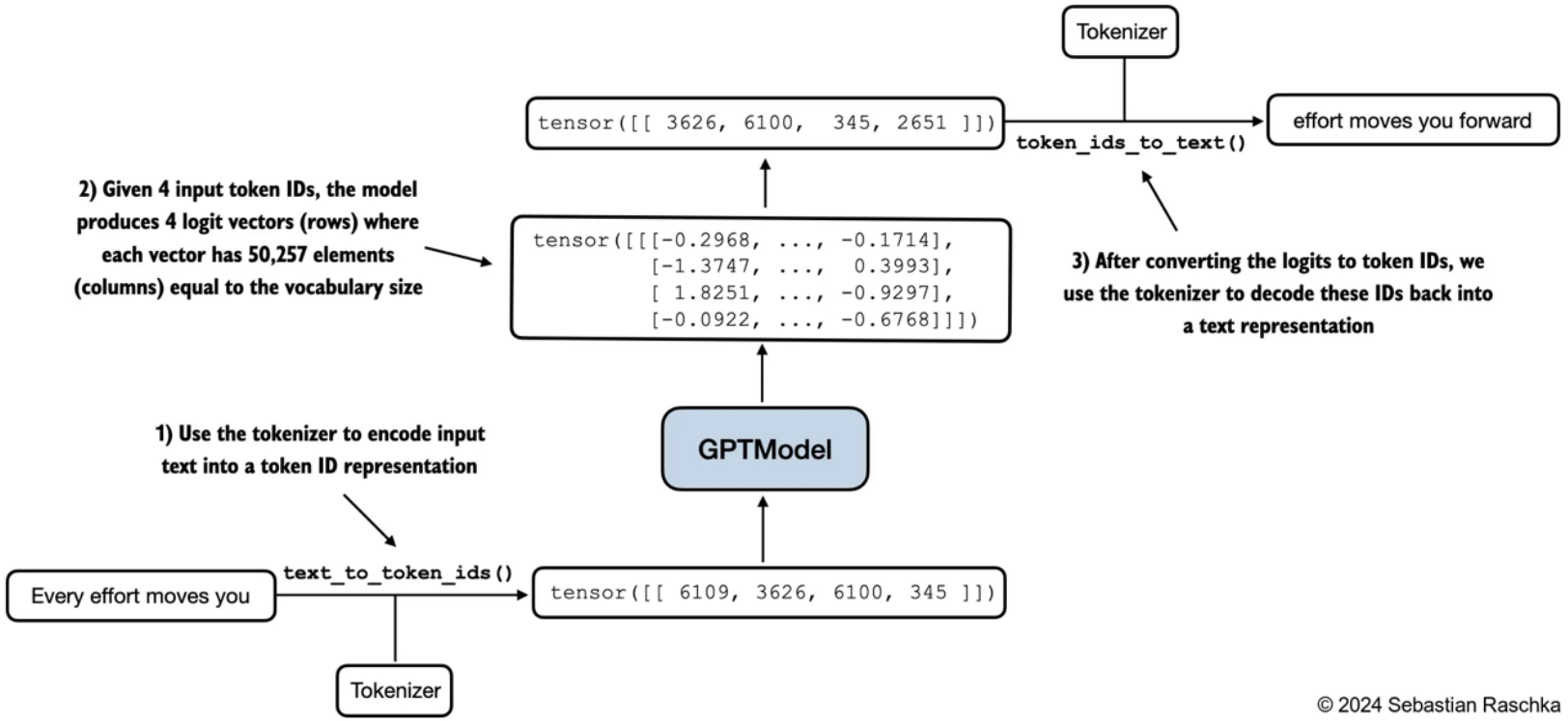
Analista de Dados e Agentes de LLM. Fonte: How Do Data Analysts Respond to AI Assistance? A Wizard-of-Oz Study.

GPT-2: Pré-treinamento

In [4]:

```
import torch
# Importamos todo o códigdo
# desenvolvido nas aulas anteriores
from llmdefinitions import GPTModel

GPT_CONFIG_124M = {
    "vocab_size": 50257,
    "context_length": 256,
    "emb_dim": 768,
    "n_heads": 12,
    "n_layers": 12,
    "drop_rate": 0.1,
    "qkv_bias": False
}

torch.manual_seed(123)
model = GPTModel(GPT_CONFIG_124M)
model.eval();
```

Usando GPT para gerar texto



A geração de texto envolve o encoding de texto para Token Ids e o processamento pelo modelo GPT com saída em forma de logits. Estes últimos são convertidos de volta a token IDs que, por sua vez, sofrem decoding para a representação textual.

In [5]:

```python
import tiktoken
from llmdefinitions import generate_text_simple

def text_to_token_ids(text, tokenizer):
    encoded = tokenizer.encode(text, allowed_special={'<|endoftext|>'})
     # Adiciona a dimensão do batch
    encoded_tensor = torch.tensor(encoded).unsqueeze(0)
    return encoded_tensor

start_context = "Every effort moves you"
tokenizer = tiktoken.get_encoding("gpt2")

token_ids = text_to_token_ids(start_context, tokenizer)

print(token_ids)
```

```
tensor([[6109, 3626, 6100,  345]])
```

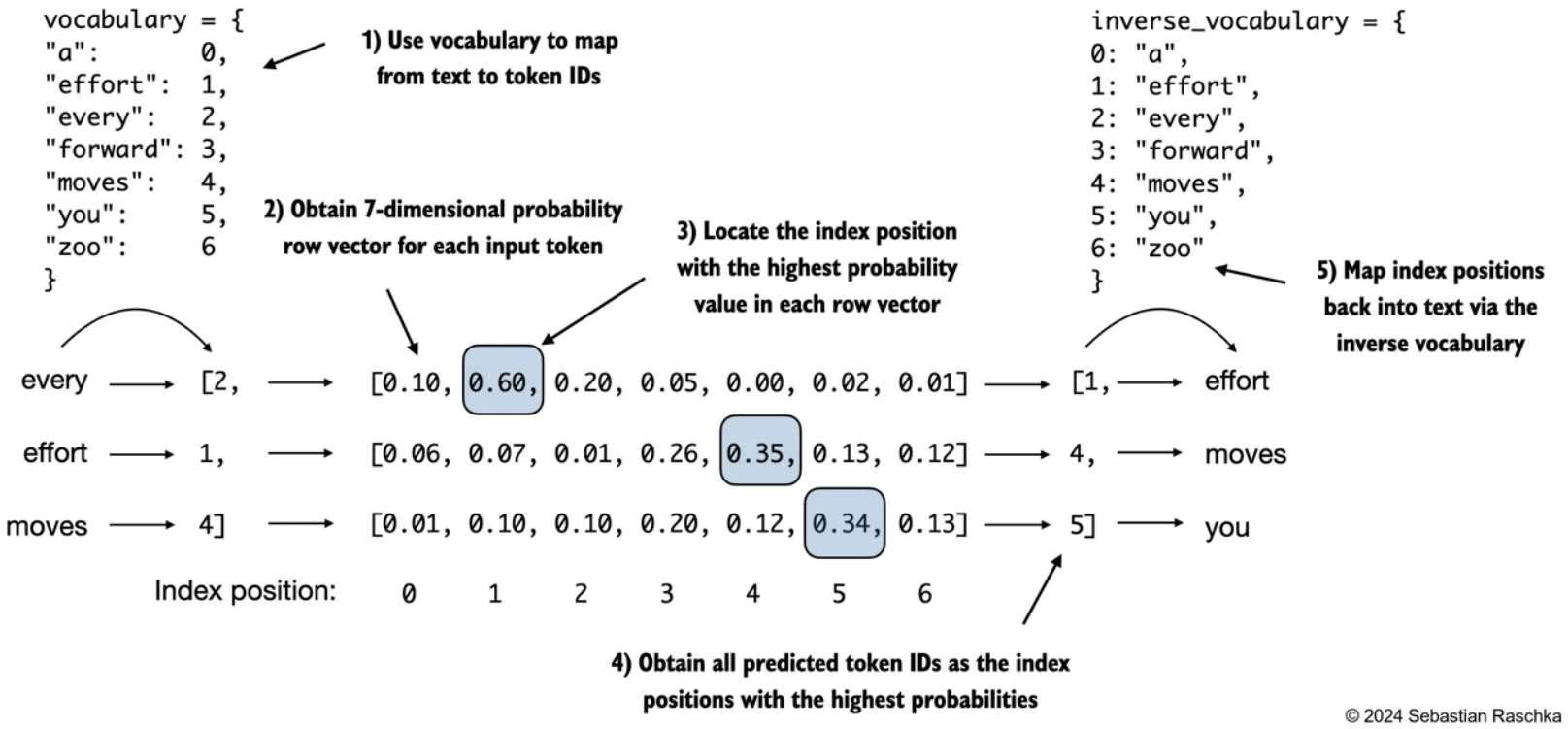
In [6]:
```
token_ids = generate_text_simple(
    model=model,
    idx=text_to_token_ids(start_context, tokenizer),
    max_new_tokens=10,
    context_size=GPT_CONFIG_124M["context_length"]
)
token_ids
```

Out[6]:
```
tensor([[ 6109,  3626,  6100,   345, 34245,  5139,  2492, 2540
5, 17434, 17853,
          5308,  3398, 13174, 43071]])
```

In [7]:
```
def token_ids_to_text(token_ids, tokenizer):
    flat = token_ids.squeeze(0) # remove batch dimension
    return tokenizer.decode(flat.tolist())

token_ids_to_text(token_ids, tokenizer)
```

Out[7]:
```
'Every effort moves you rentingetic wasn‚ refres RexMeCHicular
stren'
```

Geração de Texto



Considere os dois exemplos abaixo. Note que `targets` são as `inputs` deslocados em 1 posição. Esse deslocamento é crucial para ensinar o modelo a prever o próximo token em uma sequência.

In [8]:
```
inputs = torch.tensor(
    [[16833, 3626, 6100], # ["every effort moves",
    [40,    1107, 588]])  #  "I really like"]
targets = torch.tensor([
    [3626, 6100, 345  ],  # [" effort moves you",
    [588,  428,  11311]]) #  " really like chocolate"]
```

In [9]:
```
with torch.no_grad():
    logits = model(inputs)
print(logits.shape)
# Probabilidade para cada token no vocabulário
probas = torch.softmax(logits, dim=-1)
# Shape: (batch_size, num_tokens, vocab_size)
print(probas.shape)
token_ids = torch.argmax(probas, dim=-1, keepdim=True)
print("Token IDs:\n", token_ids)
```

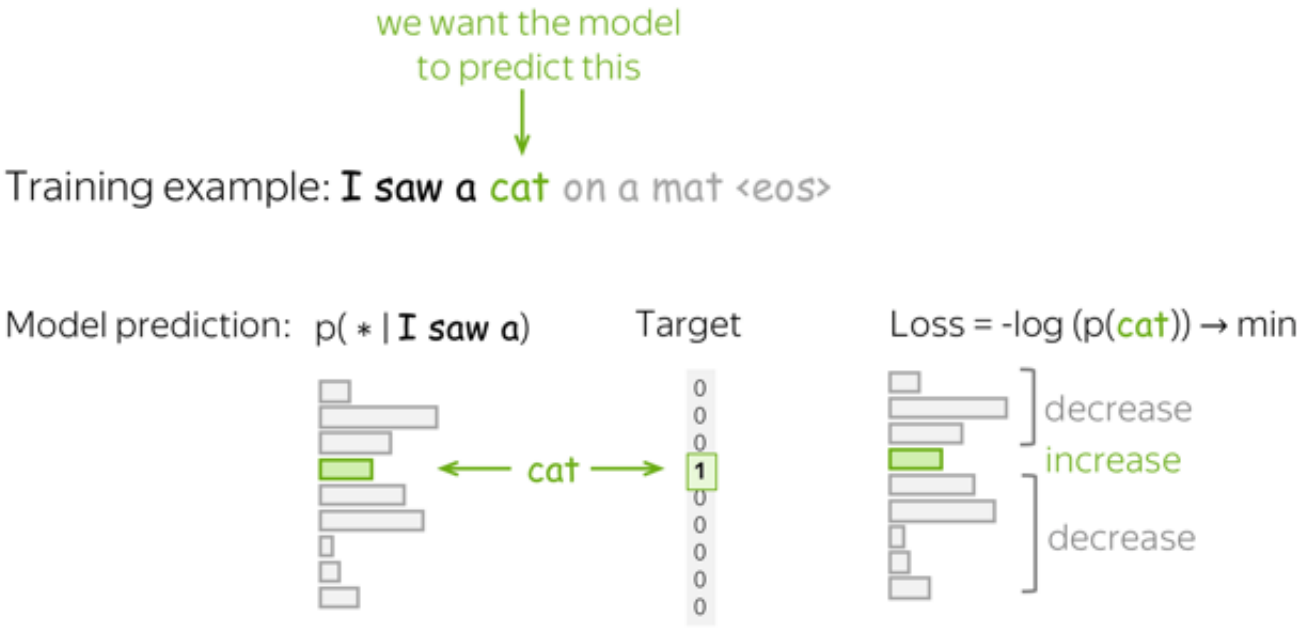
```
torch.Size([2, 3, 50257])
torch.Size([2, 3, 50257])
Token IDs:
 tensor([[[16657],
          [  339],
          [42826]],

         [[49906],
          [29669],
          [41751]]])
```

O decoding dos token IDs mostra o que foi produzido pelo modelo. Note que o como ainda não treinamos o modelo, a geração de texto é bem ruim.

In [10]:
```
print(f"Targets batch 1: {token_ids_to_text(targets[0], tokenizer)}")
out = token_ids_to_text(token_ids[0].flatten(), tokenizer)
print(f"Outputs batch 1: {out}")
```

```
Targets batch 1:  effort moves you
Outputs batch 1:  Armed heNetflix
```

Loss de geração de texto



Cross Entropy Loss. Fonte: Lena Voita.

In [11]:
```
# Logits Shape: (batch_size, num_tokens, vocab_size)
print("Logits shape:", logits.shape)
# Targets Shape: (batch_size, num_tokens)
print("Targets shape:", targets.shape)
```

```
Logits shape: torch.Size([2, 3, 50257])
Targets shape: torch.Size([2, 3])
```

In [12]:
```
logits_flat = logits.flatten(0, 1)
targets_flat = targets.flatten()
print("Flattened logits:", logits_flat.shape)
print("Flattened targets:", targets_flat.shape)
```

```
Flattened logits: torch.Size([6, 50257])
Flattened targets: torch.Size([6])
```

In [13]:
```
loss = torch.nn.functional.cross_entropy(logits_flat, targets_flat)
print(loss)
```

```
tensor(10.7722)
```

Perplexity

- Um conceito relacionado à *cross entropy loss* é a *perplexity* de um LLM, que é simplesmente a exponencial da cross entropy loss.
- A perplexidade costuma ser considerada mais interpretável porque pode ser entendida como o tamanho efetivo do vocabulário sobre o qual o modelo tem incerteza em cada passo.
- A perplexidade fornece uma medida de quão bem a distribuição de probabilidade prevista pelo modelo corresponde à distribuição real das palavras no conjunto de dados.
- Semelhante à loss, uma perplexidade menor indica que as previsões do modelo estão mais próximas da distribuição observada.

In [14]:
```
perplexity = torch.exp(loss)
print(perplexity)
```

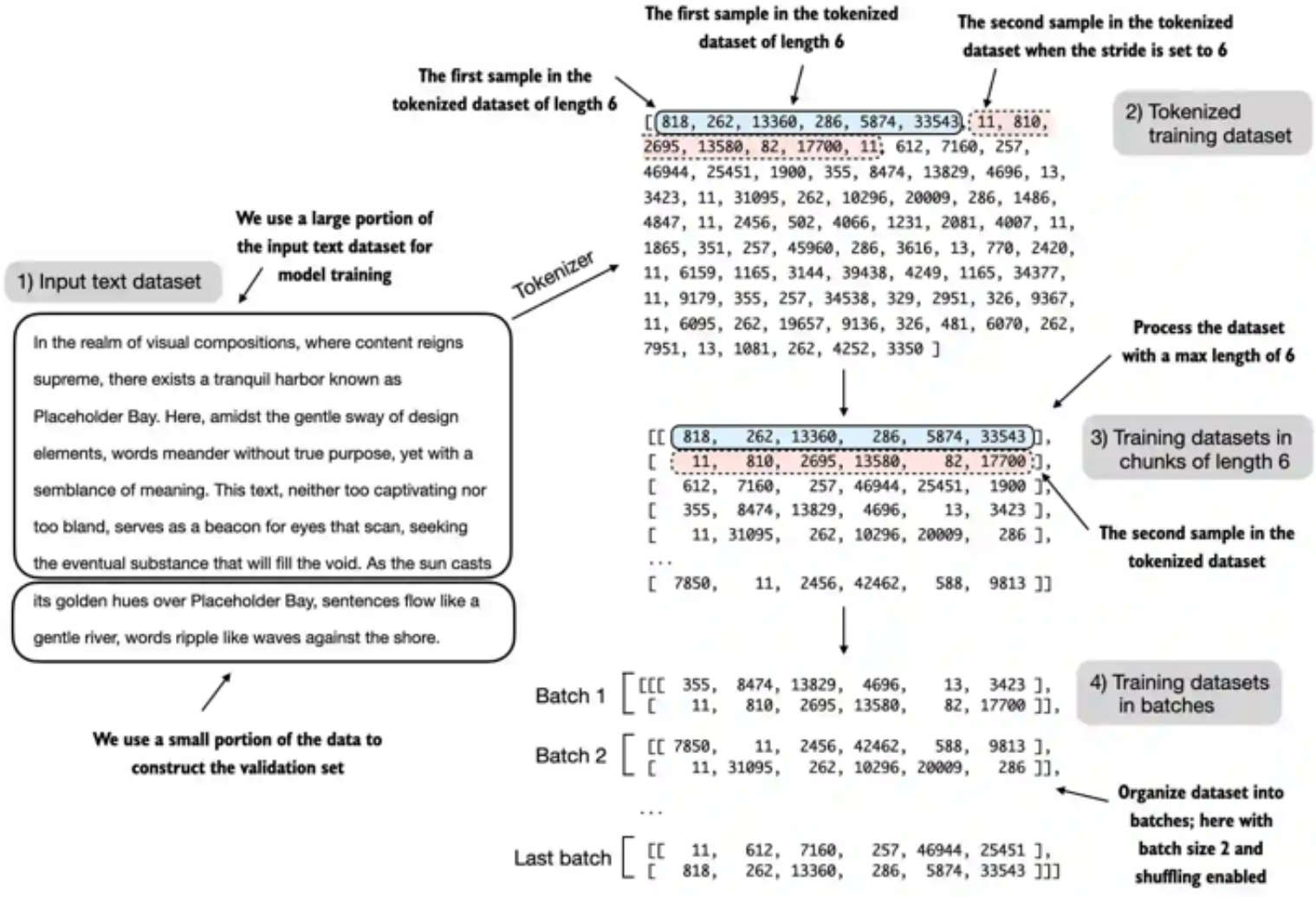
```
tensor(47678.8672)
```

Calculando loss de traino e validação

Para calcular a perda (loss) nos conjuntos de dados de treinamento e validação, utilizamos o pequeno conjunto de textos The Verdict, conto curto de Edith Wharton, com o qual já trabalhamos no início.

Curiosidade: o modelo Llama 2-7B precisou de 184.320 horas de GPU em GPUs A100 para ser treinado em 2 trilhões de tokens. Calculando o custo horário de um servidor em nuvem AWS com 8×A100 a aproximadamente 30 dóalres, treinar esse LLM custaria:
$\text{custo} = \frac{184\,320}{8} \times \$30 = \$690.000.$

Dividindo o dataset



Dividindo o dataset em treino e validação. Fonte: Sebastian Raschka.

In [17]:

```
from llmdefinitions import create_dataloader_v1

train_ratio = 0.90
split_idx = int(train_ratio * len(text_data))
train_data = text_data[:split_idx]
val_data = text_data[split_idx:]

torch.manual_seed(123)

train_loader = create_dataloader_v1(
    train_data,
    batch_size=2,
    max_length=GPT_CONFIG_124M["context_length"],
    stride=GPT_CONFIG_124M["context_length"],
    drop_last=True,
    shuffle=True,
)

val_loader = create_dataloader_v1(
    val_data,
    batch_size=2,
    max_length=GPT_CONFIG_124M["context_length"],
    stride=GPT_CONFIG_124M["context_length"],
    drop_last=False,
    shuffle=False,
)
```

Verificando os dataloaders

Temos 9 batches de treinamento com 2 amostras e 256 tokens cada um. Como alocamos apenas 10% dos dados para validação, há apenas um lote de validação composto por 2 exemplos de entrada.

In [19]:
```
print("Train loader:")
for x, y in train_loader:
    print(x.shape, y.shape)

print("\nValidation loader:")
for x, y in val_loader:
    print(x.shape, y.shape)
```

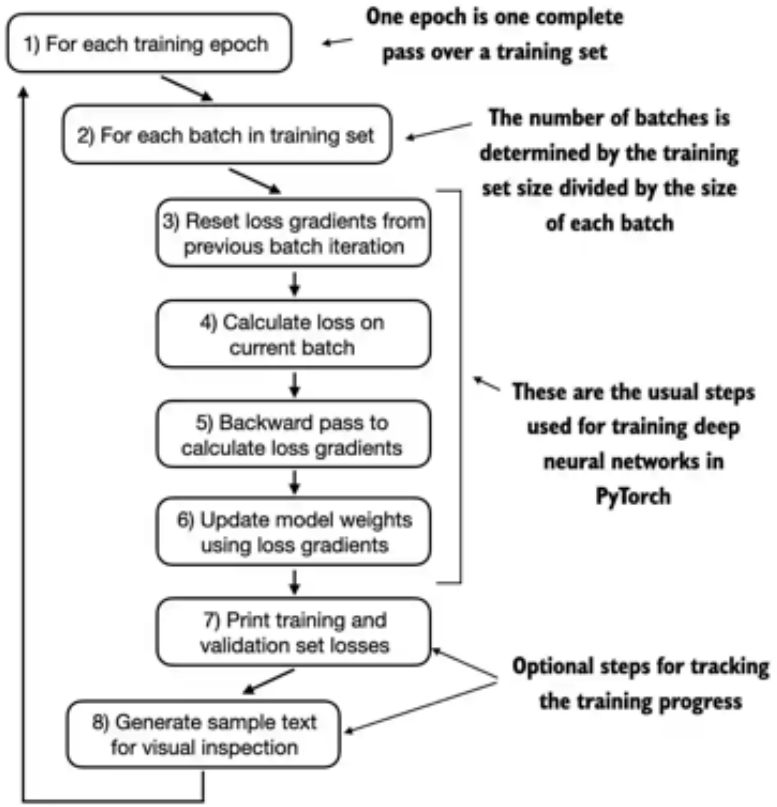
```
Train loader:
torch.Size([2, 256]) torch.Size([2, 256])
torch.Size([2, 256]) torch.Size([2, 256])
torch.Size([2, 256]) torch.Size([2, 256])
torch.Size([2, 256]) torch.Size([2, 256])
torch.Size([2, 256]) torch.Size([2, 256])
torch.Size([2, 256]) torch.Size([2, 256])
torch.Size([2, 256]) torch.Size([2, 256])
torch.Size([2, 256]) torch.Size([2, 256])
torch.Size([2, 256]) torch.Size([2, 256])

Validation loader:
torch.Size([2, 256]) torch.Size([2, 256])
```

In [23]:

```
device = set_device(False)
model.to(device)

torch.manual_seed(123)

with torch.no_grad():
    train_loss = calc_loss_loader(train_loader, model, device)
    val_loss = calc_loss_loader(val_loader, model, device)

print("Training loss:", train_loss)
print("Validation loss:", val_loss)
```

```
Training loss: 10.98758347829183
Validation loss: 10.98110580444336
```

Pré-treinamento



Pré-treinamento. Fonte: Sebastian Raschka.

In [24]:

```
def train_model_simple(model, train_loader, val_loader,
                       optimizer, device, num_epochs, eval_freq,
                       eval_iter, start_context, tokenizer):
    train_losses, val_losses, track_tokens_seen = [], [], []
    tokens_seen, global_step = 0, -1

    for epoch in range(num_epochs):
        model.train()
        for input_batch, target_batch in train_loader:
            optimizer.zero_grad()
            loss = calc_loss_batch(
                input_batch, target_batch, model, device)
            loss.backward()
            optimizer.step()
            tokens_seen += input_batch.numel()
            global_step += 1
            if global_step % eval_freq == 0:
                train_loss, val_loss = evaluate_model(
                    model, train_loader, val_loader, device, eval_iter)
                train_losses.append(train_loss)
                val_losses.append(val_loss)
                track_tokens_seen.append(tokens_seen)
                print(f"Ep {epoch+1} (Step {global_step:06d}): "
                  f"Train loss {train_loss:.3f}, Val loss {val_loss:.3f}

        generate_and_print_sample(
            model, tokenizer, device, start_context)

    return train_losses, val_losses, track_tokens_seen
```

In [25]:

```
def evaluate_model(model, train_loader, val_loader, device, eval_iter):
    model.eval()
    with torch.no_grad():
        train_loss = calc_loss_loader(
            train_loader, model,
            device, num_batches=eval_iter)
        val_loss = calc_loss_loader(
            val_loader, model,
            device, num_batches=eval_iter)
    model.train()
    return train_loss, val_loss
```

Treinando o modelo

- Treinamos o `GPTModel` por 10 épocas usando um otimizador AdamW e a função `train_model_simple` que definimos anteriormente.
- AdamW é uma variante do Adam que aprimora o método de *weight decay*, cujo objetivo é minimizar a complexidade do modelo e evitar overfitting, penalizando pesos maiores.
- Para conjuntos de dados maiores, geralmente 1 ou 2 épocas são suficientes; aqui usamos 10 porque o conjunto de treinamento é muito pequeno, permitindo que o modelo aprenda algo útil.

In [27]:
```
torch.manual_seed(123)
model = GPTModel(GPT_CONFIG_124M)
model.to(device)
optimizer = torch.optim.AdamW(model.parameters(),
                              lr=0.0004, weight_decay=0.1)
num_epochs = 10
train_losses, val_losses, tokens_seen = train_model_simple(
    model, train_loader, val_loader, optimizer, device,
    num_epochs=num_epochs, eval_freq=5, eval_iter=5,
    start_context="Every effort moves you", tokenizer=tokenizer
)
```

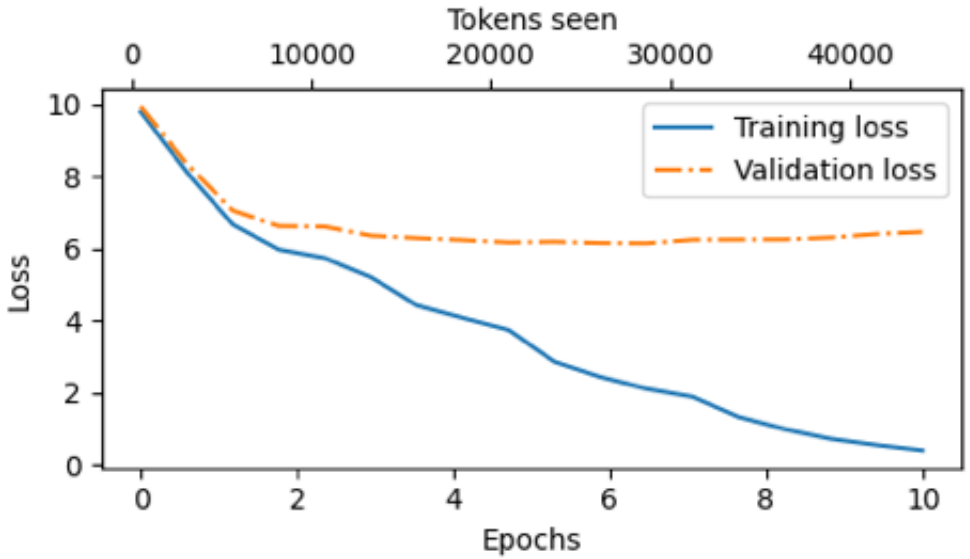
```
Ep 1 (Step 000000): Train loss 9.781, Val loss 9.933
Ep 1 (Step 000005): Train loss 8.111, Val loss 8.339
Every effort moves you,,,,,,,,,,,,.
Ep 2 (Step 000010): Train loss 6.661, Val loss 7.048
```

```
Ep 2 (Step 000015): Train loss 5.961, Val loss 6.616
Every effort moves you, and, and, and, and, and, and, and, and,
and, and, and, and, and, and, and, and, and, and, and, and, and,
and,, and, and,
Ep 3 (Step 000020): Train loss 5.726, Val loss 6.600
Ep 3 (Step 000025): Train loss 5.201, Val loss 6.348
Every effort moves you, and I had been.
Ep 4 (Step 000030): Train loss 4.417, Val loss 6.278
Ep 4 (Step 000035): Train loss 4.069, Val loss 6.226
Every effort moves you know the                          "I he h
ad the donkey and I had the and I had the donkey and down the ro
om, I had
Ep 5 (Step 000040): Train loss 3.732, Val loss 6.160
Every effort moves you know it was not that the picture--I had t
he fact by the last I had been--his, and in the            "Oh,
and he said, and down the room, and in
Ep 6 (Step 000045): Train loss 2.850, Val loss 6.179
Ep 6 (Step 000050): Train loss 2.427, Val loss 6.141
Every effort moves you know," was one of the picture. The--I had
a little of a little: "Yes, and in fact, and in the picture was,
and I had been at my elbow and as his pictures, and down the roo
m, I had
Ep 7 (Step 000055): Train loss 2.104, Val loss 6.134
Ep 7 (Step 000060): Train loss 1.882, Val loss 6.233
Every effort moves you know," was one of the picture for nothing
--I told Mrs.  "I was no--as! The women had been, in the moment-
-as Jack himself, as once one had been the donkey, and were, and
in his
Ep 8 (Step 000065): Train loss 1.320, Val loss 6.238
Ep 8 (Step 000070): Train loss 0.985, Val loss 6.242
Every effort moves you know," was one of the axioms he had been
the tips of a self-confident moustache, I felt to see a smile be
hind his close grayish beard--as if he had the donkey. "stronges
```

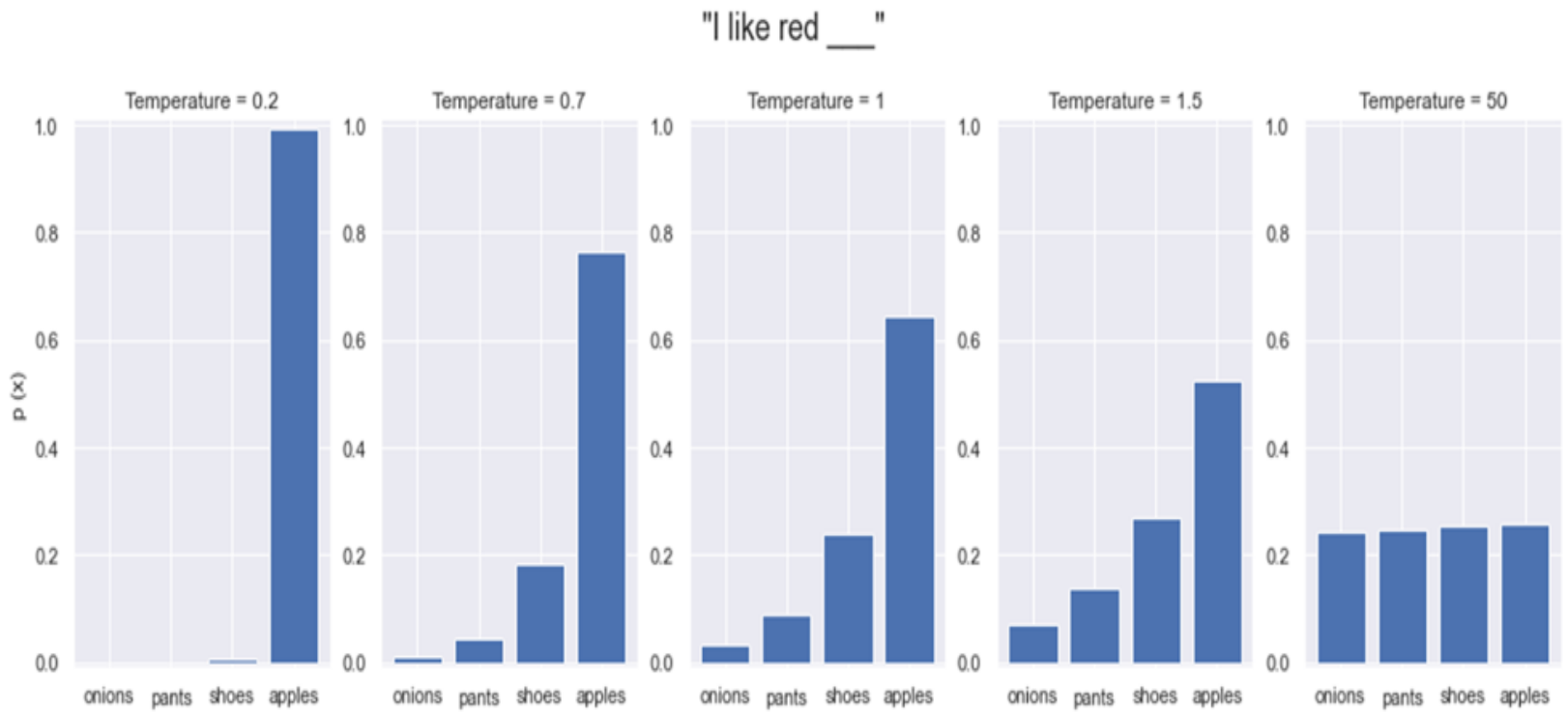
t," as his
Ep 9 (Step 000075): Train loss 0.717, Val loss 6.293
Ep 9 (Step 000080): Train loss 0.541, Val loss 6.393
Every effort moves you?" "Yes--quite insensible to the irony. She wanted him vindicated--and by me!" He laughed again, and threw back the window-curtains, I had the donkey. "There were days when I
Ep 10 (Step 000085): Train loss 0.391, Val loss 6.452
Every effort moves you know," was one of the axioms he laid down across the Sevres and silver of an exquisitely appointed luncheon-table, when, on a later day, I had again run over from Monte Carlo; and Mrs. Gis

In [29]:

```
epochs_tensor = torch.linspace(0, num_epochs, len(train_losses))
plot_losses(epochs_tensor, tokens_seen, train_losses, val_losses)
```

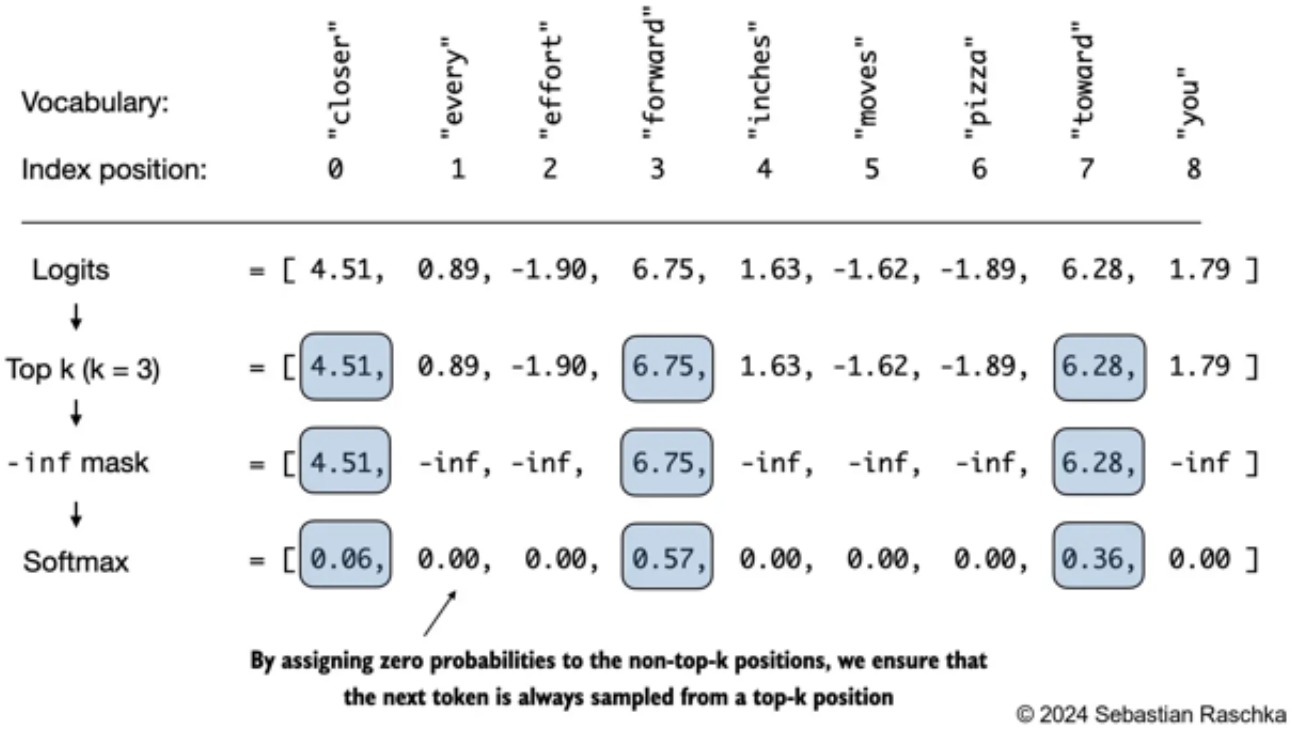


Temperature scaling



Temperature Scaling. Fonte: Hopsworks.ai.

Top-k sampling



Top-k sampling. Fonte: Sebastian Raschka.

Combinando estratégias

In [30]:

```python
def generate(model, idx, max_new_tokens, context_size, temperature, top_
    for _ in range(max_new_tokens):
        idx_cond = idx[:, -context_size:]
        with torch.no_grad():
            logits = model(idx_cond)
        logits = logits[:, -1, :]
        # Aplica top-k sampling
        if top_k is not None:
            top_logits, _ = torch.topk(logits, top_k)
            min_val = top_logits[:, -1]
            logits = torch.where(
                logits < min_val,
                torch.tensor(float('-inf')).to(logits.device), logits)
        # Aplica temperature scaling
        if temperature > 0.0:
            logits = logits / temperature
            probs = torch.softmax(logits, dim=-1)
            idx_next = torch.multinomial(probs, num_samples=1)
        else:
            idx_next = torch.argmax(logits, dim=-1, keepdim=True)

        idx = torch.cat((idx, idx_next), dim=1)

    return idx
```

In [31]:

```
torch.manual_seed(123)

token_ids = generate(
    model=model,
    idx=text_to_token_ids("Every effort moves you", tokenizer),
    max_new_tokens=15,
    context_size=GPT_CONFIG_124M["context_length"],
    top_k=25,
    temperature=1.4
)

print("Output text:\n", token_ids_to_text(token_ids, tokenizer))
```

```
Output text:
 Every effort moves youlit terrace.



" he said deprecating laugh
```