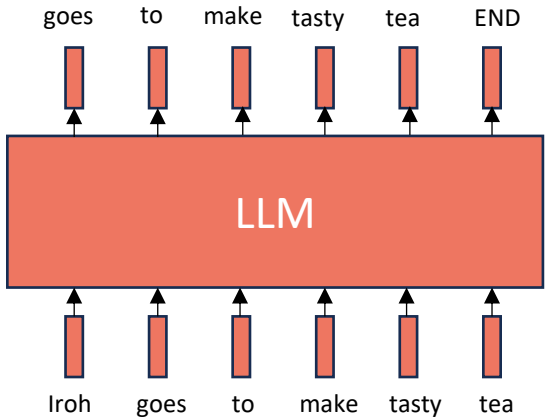
Lecture10
Pre-entrenamiento y Alineación de LLMs

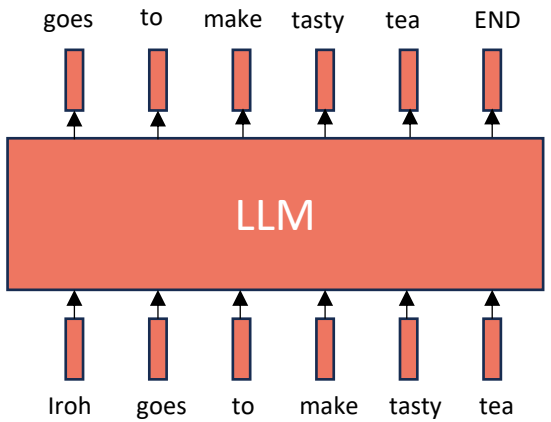
- Pre-entrenamiento
- Sintonización supervisada
- Alineación de IA
- Aprendizaje por refuerzo
  - Configuración de RL
  - Conexión entre RL y modelos de lenguaje
  - Modelo de recompensa
  - Modelo de Bradley-Terry
  - Función objetivo en RLHF
- DPO
  - Función de pérdida
  - Cálculo de las probabilidades logarítmicas

Pre-entrenamiento

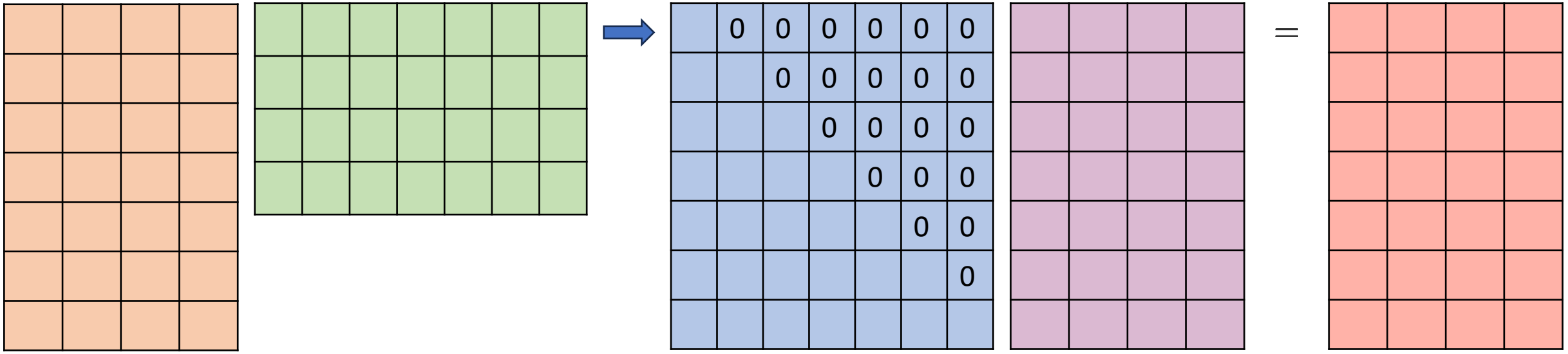
- Modelar $p_\theta(w_t|w_{1:t-1})$ la distribución de probabilidad sobre palabras dado su contexto anterior.
- ¡Hay muchos datos para esto! (¡En inglés!)
- **Pre-entrenamiento mediante modelado del lenguaje**:
- Entrenar una red neuronal para realizar el modelado del lenguaje en una gran cantidad de texto.
- Guardar los parámetros de la red.



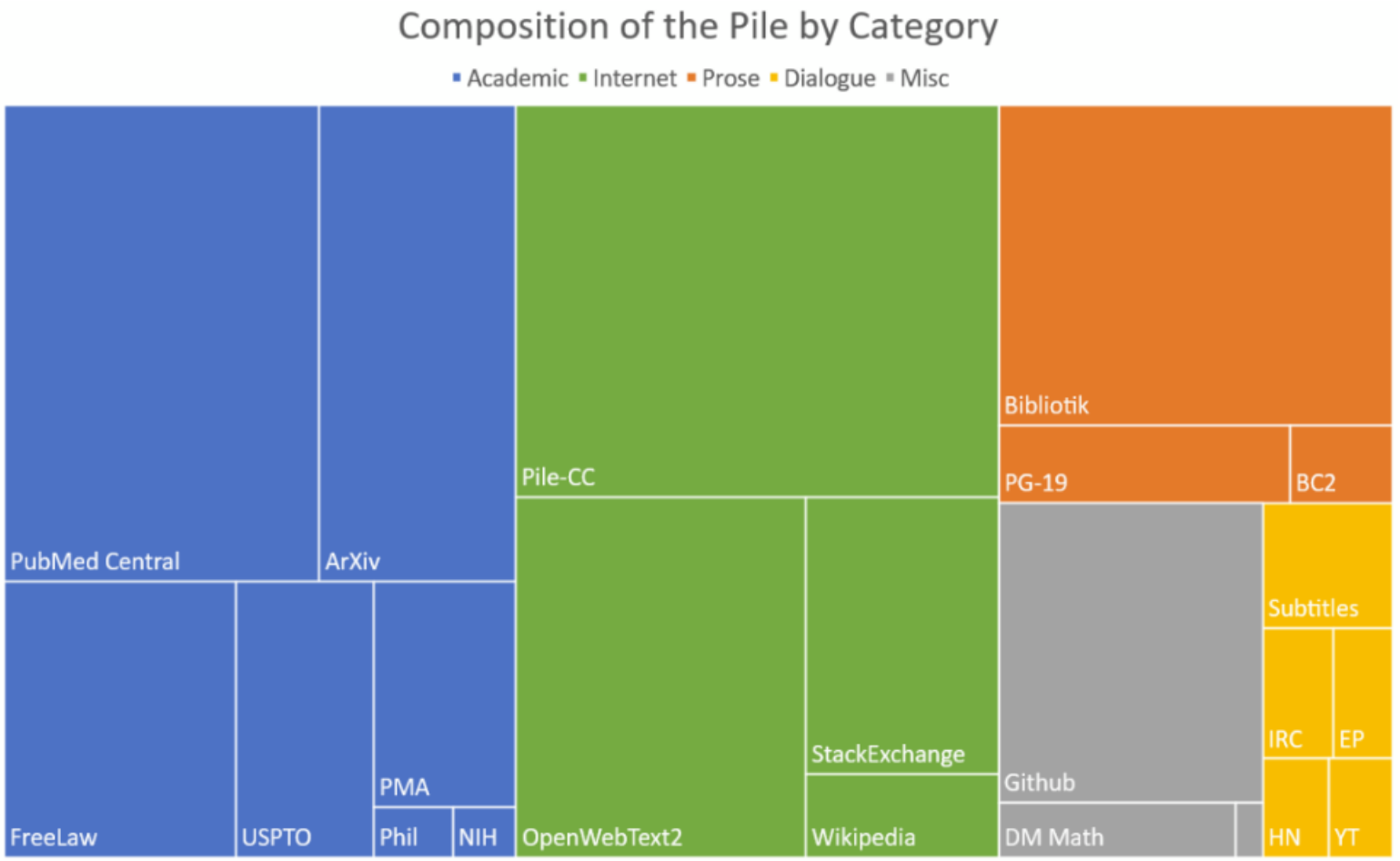
Pre-entrenamiento



$\mathbf{Q}_{qa}$ $\mathbf{K}^\top$ $\text{softmax}\left(\mathbf{Q}_{qa}\mathbf{K}^\top/\sqrt{(d_k)}\right)$ $\mathbf{V}$ $\mathbf{A}$



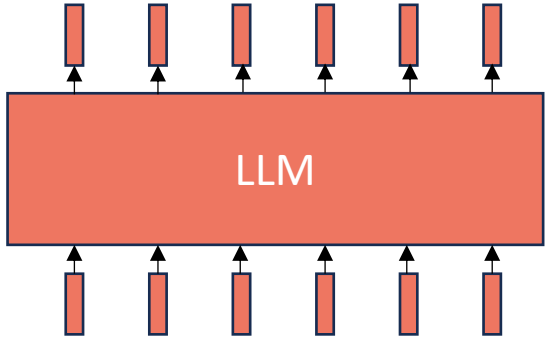
¿De dónde vienen los datos?

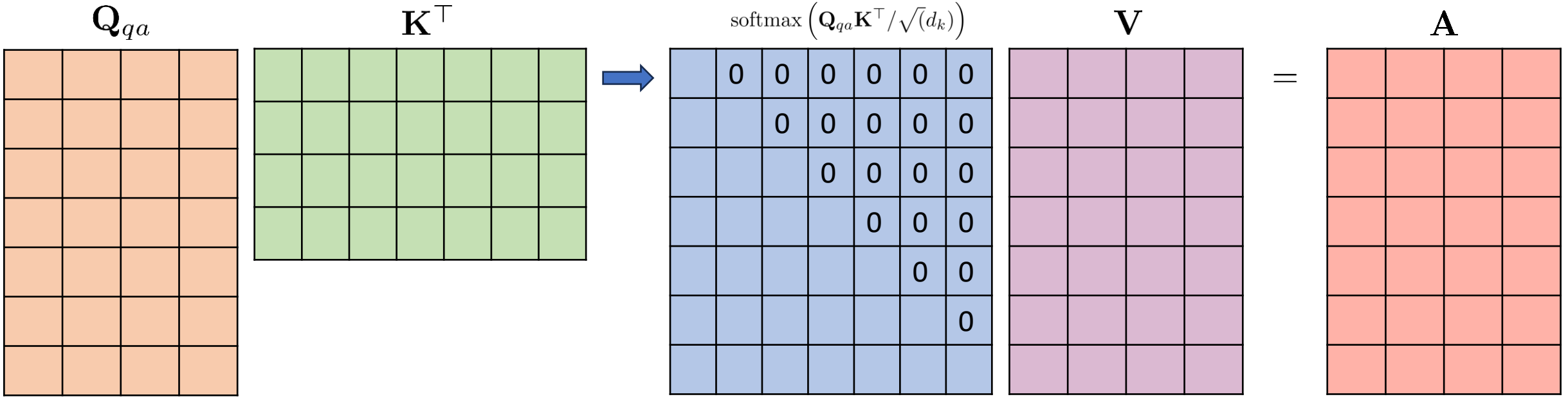


Model	Training Data
BERT	BookCorpus, English Wikipedia
GPT-1	BookCorpus
GPT-3	CommonCrawl, WebText, English Wikipedia, and 2 book databases (“Books 1” and “Books 2”)
GPT-3.5+	Undisclosed

Función de costo

Sintonización supervisada





Modelo de lenguaje

THE COST OF TRAINING NLP MODELS
A CONCISE OVERVIEW

Or Sharir
AI21 Labs
ors@ai21.com

Barak Peleg
AI21 Labs
barakp@ai21.com

Yoav Shoham
AI21 Labs
yoavs@ai21.com

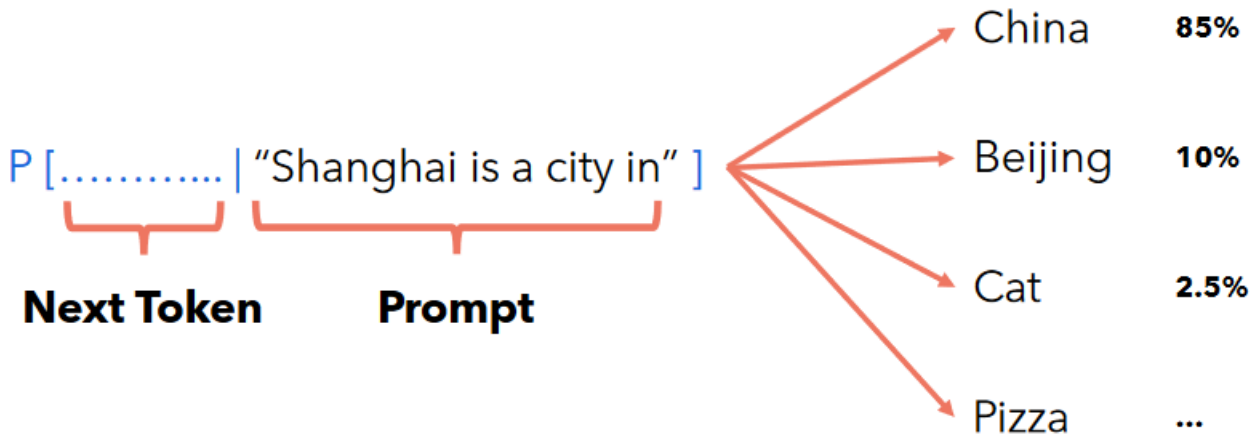
April 2020

Costos: No aptos para los débiles de corazón

- $ 2.5k - $ 50k (modelo de 110 millones de parámetros)
- $ 10k - $ 200k (340 millones)
- $ 80k - $ 1.6m (1.500 millones)

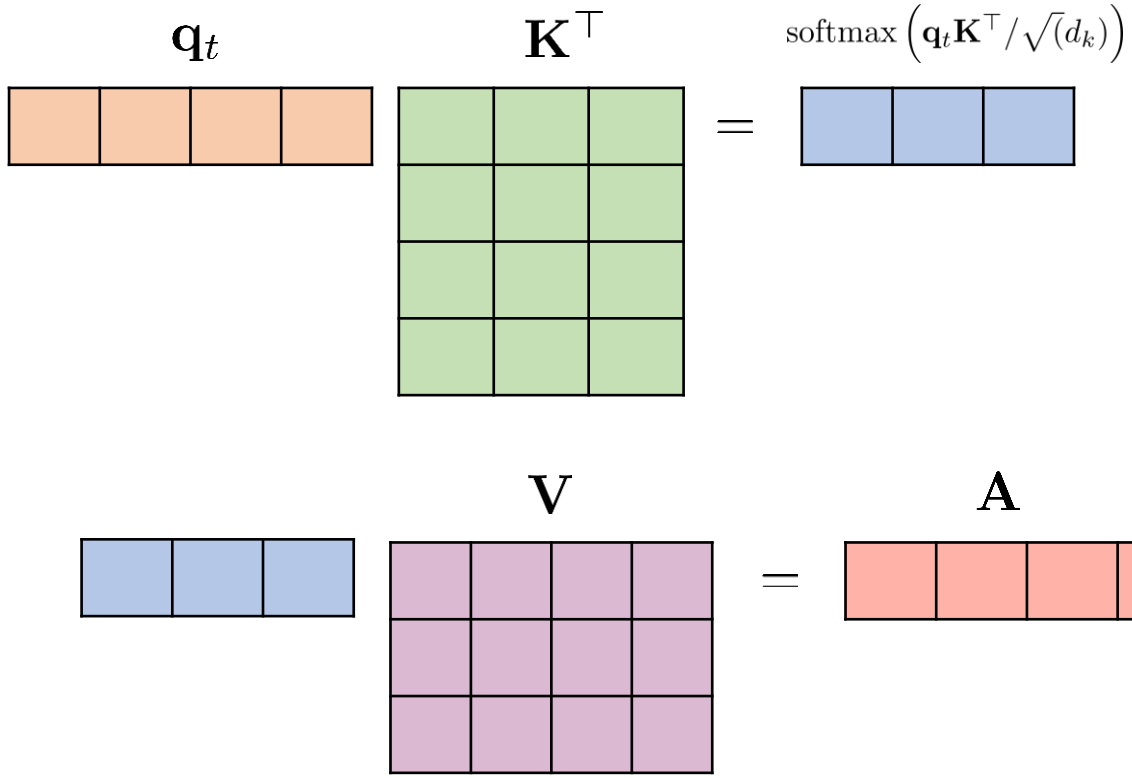
Modelo de lenguaje

Un modelo de lenguaje es un modelo probabilístico que asigna probabilidades a secuencias de palabras. En la práctica, un modelo de lenguaje nos permite calcular lo siguiente:

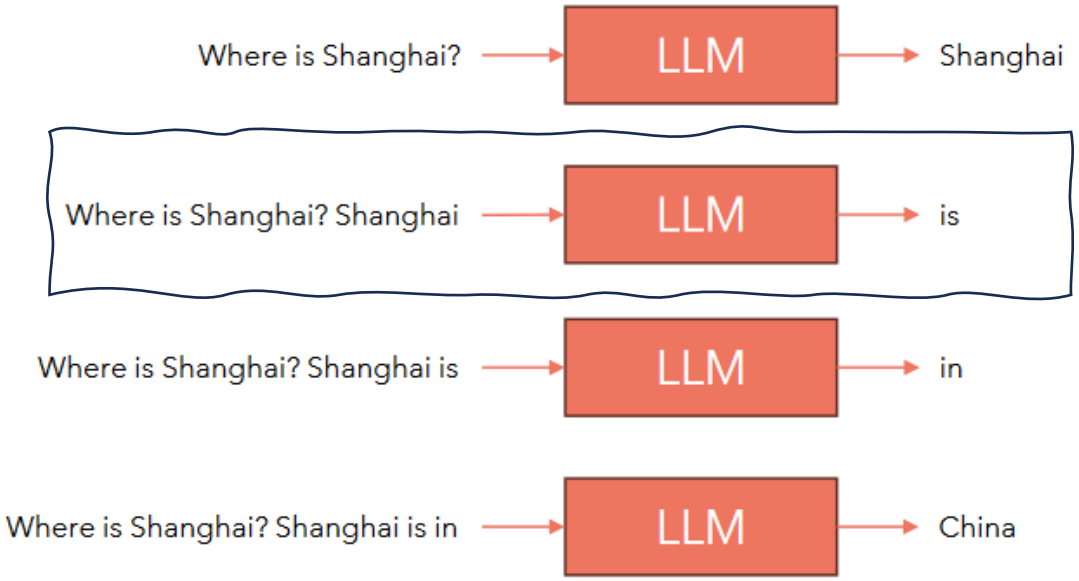


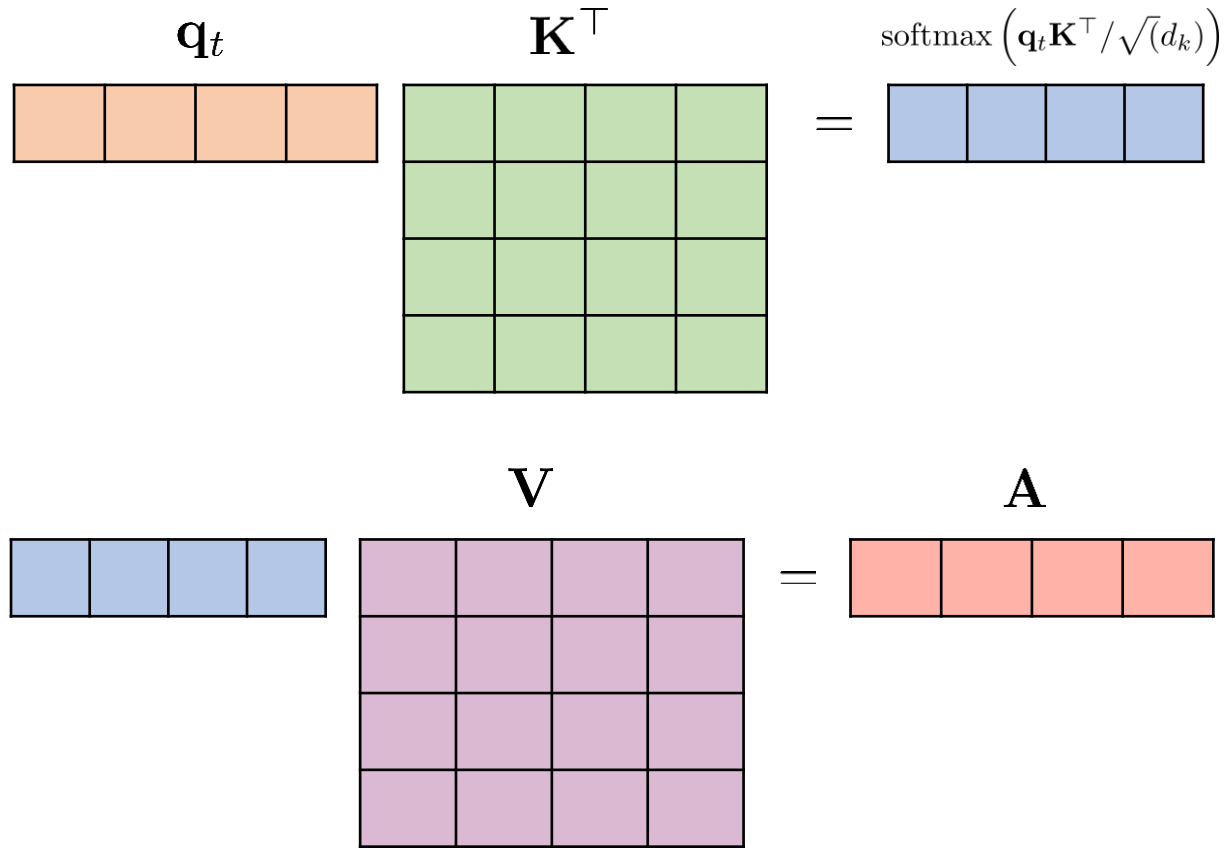
Un modelo de lenguaje genera una lista de probabilidades, una para cada token en el vocabulario, indicando qué tan probable es que un token sea el siguiente.

Generando de forma iterativa el siguiente token



Generando de forma iterativa el siguiente token





Alineación de IA

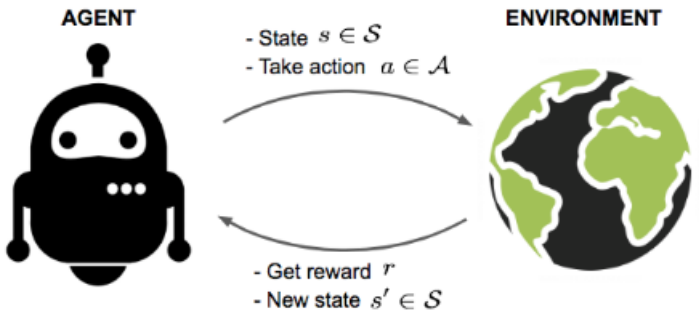
Un modelo de lenguaje grande típicamente se pre-entrena con una enorme cantidad de datos, por ejemplo, toda la Wikipedia y miles de millones de páginas web. Esto le da al modelo de lenguaje un vasto “conocimiento” de información para completar cualquier solicitud de manera razonable. Sin embargo, para usar un LLM como un asistente de chat (por ejemplo, ChatGPT), queremos forzar al modelo de lenguaje a seguir un estilo particular. Por ejemplo, podríamos querer lo siguiente:

- No usar lenguaje ofensivo
- No usar expresiones racistas
- Responder preguntas utilizando un estilo particular

El objetivo de la alineación de IA es alinear el comportamiento del modelo con un comportamiento deseado.

Introducción a aprendizaje por refuerzo

El Aprendizaje por Refuerzo se ocupa de cómo un agente inteligente debe tomar acciones en un entorno para maximizar la recompensa acumulada.



Configuración de RL

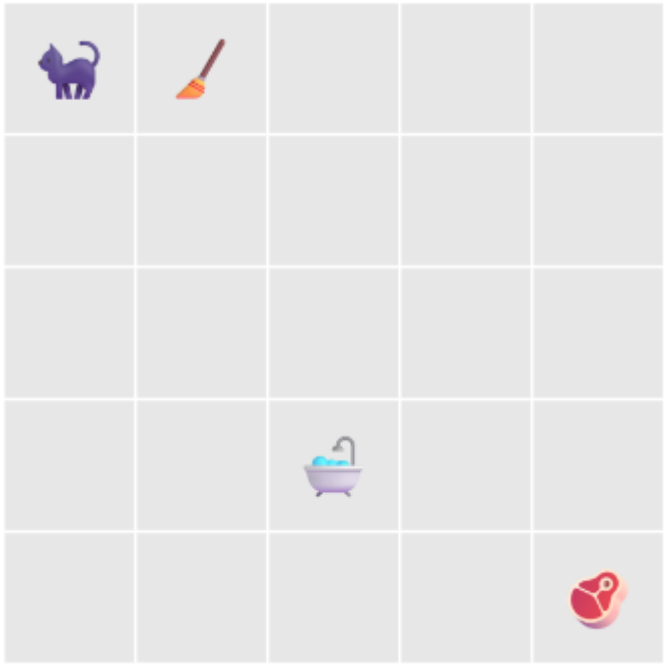
Agente: el gato
Estado: la posición del gato (x, y) en la cuadrícula
Acción: en cada posición, el gato puede moverse a una de las celdas conectadas en las 4 direcciones. Si un movimiento es inválido, la celda no se moverá y permanecerá en la misma posición. Cada vez que el gato realiza un movimiento, esto resulta en un nuevo estado y una recompensa.
Modelo de recompensa:

- Un movimiento hacia otra celda vacía resulta en una recompensa de 0.
- Un movimiento hacia la escoba resultará en una recompensa de -1.
- Un movimiento hacia la bañera resultará en una recompensa de -10 y el desmayo del gato (fin del episodio). El gato será reubicado en la posición inicial nuevamente.
- Un movimiento hacia la carne resultará en una recompensa de +100.

Política: una política dicta cómo el agente selecciona la acción a realizar dado el estado en el que se encuentra:

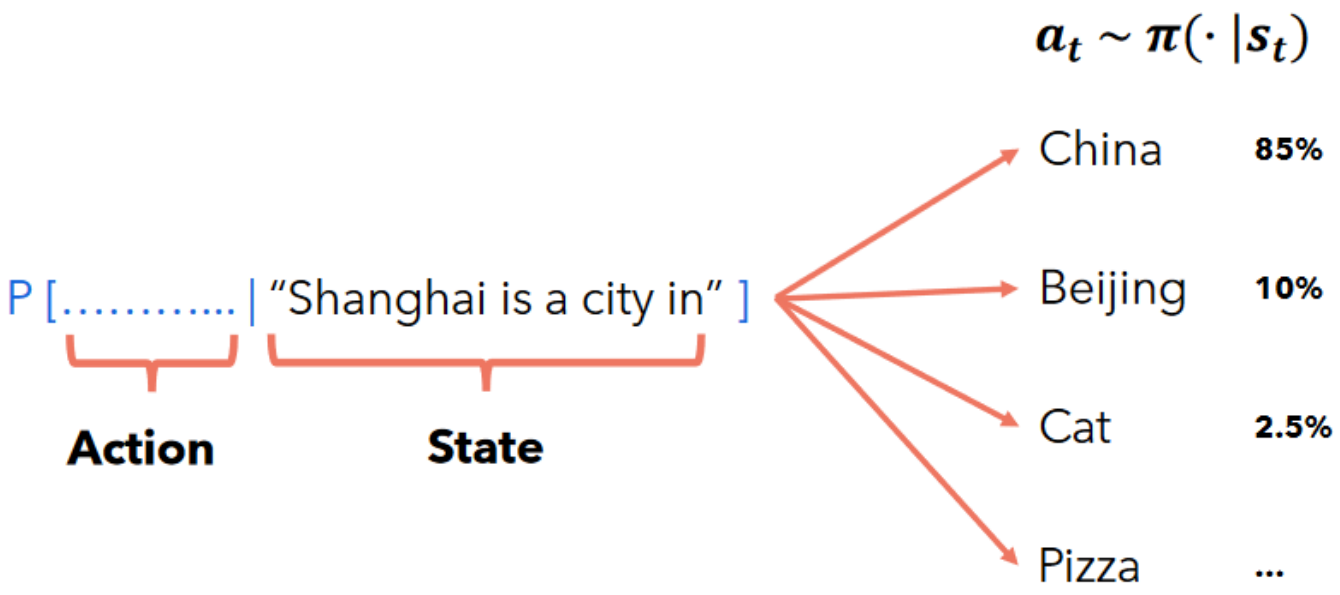
$$a_t \sim \pi(\cdot \mid s_t)$$

El objetivo en RL es seleccionar una política que maximice el retorno esperado cuando el agente actúa de acuerdo con ella.



Configuración de RL: Conexión con LLMs

Agente: El modelo de lenguaje
Estado: El prompt (tokens de entrada)
Acción: Seleccionar el siguiente token
Modelo de recompensa: El modelo debería ser recompensado por generar “buenas respuestas” y no debería ser recompensado por generar “malas respuestas”
Política: En el caso del modelo de lenguaje, la política es el mismo modelo de lenguaje dado que este modela la probabilidad del espacio de acción dado el estado actual del agente



Modelo de recompensa para los LLMs

No es fácil crear un modelo de recompensa para los modelos de lenguaje, porque esto requeriría que creáramos un conjunto de datos de indicaciones y respuestas, y asignáramos una “recompensa” universalmente aceptada para cada respuesta.

Question (Prompt)	Answer (Text generated by the language model)	Reward (0.0 ~ 1.0)
Where is Shanghai?	Shanghai is a city in China	???
Explain gravity like I’m 5	Gravity is what pulls things toward each other. It's why you stay on the ground and planets orbit the sun.	???
What is 2+2?	4	???

No somos buenos generando una referencia común, pero somos buenos comparando.

Modelo de recompensa por comparación

Base de datos de preguntas y respuestas y personas seleccionan cuál prefieren.

Question (Prompt)	Answer 1	Answer 2	Chosen
Where is Shanghai?	Shanghai is a city in China	Shanghai does not exist	1
Explain gravity like I'm 5	Gravity is a famous restaurant	Gravity is what pulls things toward each other. It's why you stay on the ground and planets orbit the sun.	2
What is 2+2?	4	2+2 is a very complicated math problem…	1

$$P(y_w > y_l) = \frac{e^{r^*(x, y_w)}}{e^{r^*(x, y_w)} + e^{r^*(x, y_l)}}$$

El modelo se ajusta sobre esta base de datos

Modelo Bradley-Terry

Ahora que tenemos un conjunto de datos de preferencias, necesitamos encontrar una manera de convertir esas preferencias en una puntuación (recompensa). Una forma de hacerlo es modelar nuestras preferencias utilizando el modelo de Bradley-Terry.

$$P(y_w > y_l) = \frac{e^{r^*(x,y_w)}}{e^{r^*(x,y_w)} + e^{r^*(x,y_l)}}$$

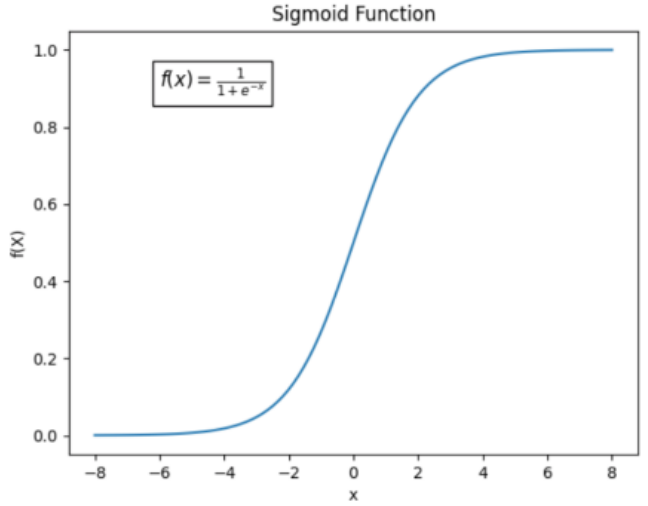
x y_w y_l

Question/Prompt (x)	Good/winning answer (y_w)	Bad/losing answer (y_l)
Where is Shanghai?	Shanghai is a city in China	Shanghai does not exist
Explain gravity like I'm 5	Gravity is what pulls things toward each other. It's why you stay on the ground and planets orbit the sun.	Gravity is a famous restaurant
What is 2+2?	4	2+2 is a very complicated math problem…

Función de costo del modelo de recompensa

$$P(y_w > y_l) = \frac{e^{r^\phi(x,y_w)}}{e^{r^\phi(x,y_w)} + e^{r^\phi(x,y_l)}}$$

$$\frac{e^A}{e^A + e^B} \implies \sigma(A - B)$$



$$\frac{e^A}{e^A + e^B} = \frac{e^A}{e^A\left(\frac{e^A+e^B}{e^A}\right)} = \frac{1}{\frac{e^A+e^B}{e^A}} = \frac{1}{1+\frac{e^B}{e^A}} = \frac{1}{1+e^{B-A}} = \sigma(A - B)$$

Esto nos da la expresión para el modelo de recompensa en DPO

$$L = -\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log \sigma\left(r_\phi(x, y_w) - r_\phi(x, y_l)\right)\right]$$

Función de costo RLHF

$$J_{\text{RLHF}} = \max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)} \left[r_\phi(x,y) - \beta \mathcal{D}_{KL} \left[\pi_\theta(y|x) \| \pi_{\text{ref}}(y|x) \right] \right]$$

Maximizar la recompensa

Restringir el modelo para que no sea tan diferente del modelo de referencia

Política óptima

Según el artículo DPO, el problema de optimización tiene una solución cerrada:

$$\pi_r(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)$$

$$Z(x) = \sum_y \pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)$$

Sin embargo, el término de normalización no es tratable computacionalmente porque deberíamos generar todas las posibles respuestas y que puede generar nuestro modelo de lenguaje para cada prompt x.

Política óptima

Suponga que, de alguna forma, tenemos acceso a la política óptima. ¿Cómo sería la función de recompensa para esa política?

$$\log \pi^*(y|x) = \log \left[\frac{1}{Z(x)} \pi_{\text{ref}}(y|x) \exp\left(\frac{1}{\beta} r(x,y) \right) \right]$$

$$= \log \pi_{\text{ref}}(y|x) - \log Z(x) + \log \exp\left(\frac{1}{\beta} r(x,y) \right)$$

$$= \log \pi_{\text{ref}}(y|x) - \log Z(x) + \frac{1}{\beta} r(x,y)$$

$$r(x,y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)$$

Si tuviéramos la política óptima, podríamos utilizarla para obtener la función de recompensa. ¿Qué hacemos con esto?

… retomemos Bradley-Terry

$$P(y_w > y_l) = \frac{e^{r^\phi(x,y_w)}}{e^{r^\phi(x,y_w)} + e^{r^\phi(x,y_l)}} \qquad \frac{e^A}{e^A + e^B} \implies \sigma(A - B)$$

Ubiquemos la expresión de la función de recompensa en Bradley-Terry

$$r(x, y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)$$

$$P(y_w > y_l) = \sigma\left(r(x, y_w) - r(x, y_l)\right) = \sigma\left(\beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} + \beta \log Z(x) - \beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)} - \beta \log Z(x)\right)$$

Para aprender un modelo de Bradley-Terry que maximice la probabilidad de elegir y_w sobre y_l, solo necesitamos maximizar la expresión anterior o minimizar la expresión negativa a continuación.

$$L_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right]$$

… retomemos Bradley-Terry

Para aprender un modelo de Bradley-Terry que maximice la probabilidad de elegir y_w sobre y_l, solo necesitamos maximizar la expresión anterior o minimizar la expresión negativa a continuación.

$$L_{\mathrm{DPO}}(\pi_\theta; \pi_{\mathrm{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\mathrm{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\mathrm{ref}}(y_l|x)} \right) \right]$$

Así que, en lugar de optimizar la función de recompensa, estamos optimizando la política óptima (que depende de la función de recompensa). ¡Una solución brillante!

¿Cómo calculamos las log-probs?

Para evaluar la pérdida, necesitamos calcular las log-probs en la expresión:

$$L_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right]$$

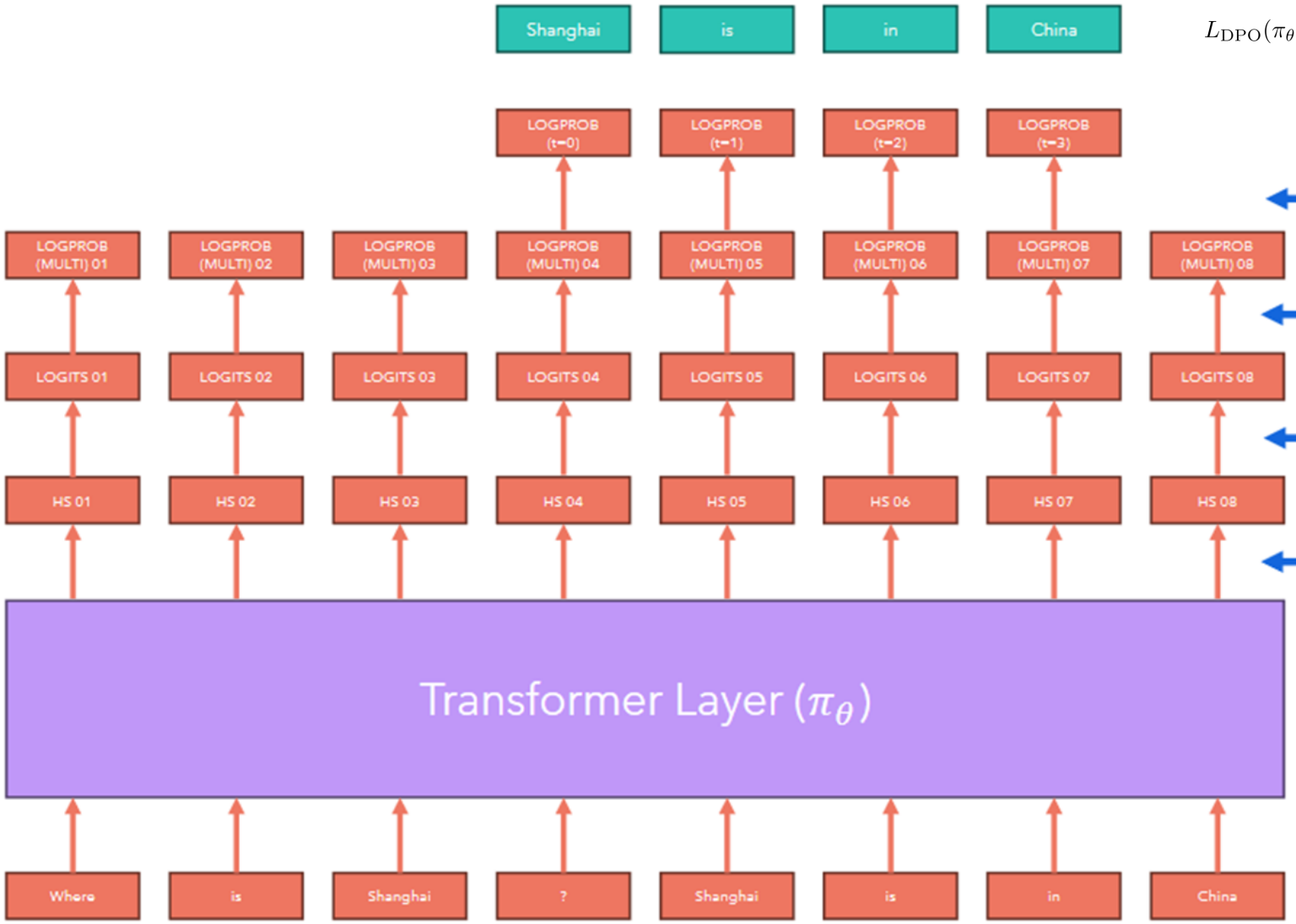
```
dpo_dataset_dict = {
    "prompt": [
        "hello",
        "how are you",
        "What is your name?",
        "What is your name?",
        "Which is the best programming language?",
        "Which is the best programming language?",
        "Which is the best programming language?",
    ],
    "chosen": [
        "hi nice to meet you",
        "I am fine",
        "My name is Mary",
        "My name is Mary",
        "Python",
        "Python",
        "Java",
    ],
    "rejected": [
        "leave me alone",
        "I am not fine",
        "Whats it to you?",
        "I dont have a name",
        "Javascript",
        "C++",
        "C++",
    ],
}
```

```
training_args = DPOConfig(
    beta=0.1,
)
dpo_trainer = DPOTrainer(
    model,
    ref_model,
    args=training_args,
    train_dataset=train_dataset,
    tokenizer=tokenizer,  # for visual language models, use tokenizer=processor instead
)
```

```
dpo_trainer.train()
```

¿Cómo calculamos las log-probs?

$$L_{\mathrm{DPO}}(\pi_\theta;\pi_{\mathrm{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\mathrm{ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\mathrm{ref}}(y_l|x)}\right)\right]$$



Seleccionar el log_prob correspondiente al token siguiente

Aplicar log_softmax para generar log probs

Se aplica una proyección lineal a todos los estados ocultos para generar logits en todas las posiciones

Cada estado oculto codifica información de los tokens previos (enmascaramiento causal)

¿Cómo calculamos las log-probs?

$$L_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right]$$

```python
@staticmethod
def get_batch_logps(
    logits: torch.FloatTensor,
    labels: torch.LongTensor,
    average_log_prob: bool = False,
    label_pad_token_id: int = -100,
    is_encoder_decoder: bool = False,
) -> torch.FloatTensor:
    """Compute the log probabilities of the given labels under the given logits.

    Args:
        logits: Logits of the model (unnormalized). Shape: (batch_size, sequence_length, vocab_size)
        labels: Labels for which to compute the log probabilities. Label tokens with a value of label_pad
        average_log_prob: If True, return the average log probability per (non-masked) token. Otherwise,
        label_pad_token_id: The label pad token id.
        is_encoder_decoder: Whether the model is an encoder-decoder model.

    Returns:
        A tensor of shape (batch_size,) containing the average/sum log probabilities of the given labels
    """
    if logits.shape[:-1] != labels.shape:
        raise ValueError("Logits (batch and sequence length dim) and labels must have the same shape.")

    if not is_encoder_decoder:
        labels = labels[:, 1:].clone()
        logits = logits[:, :-1, :]
    loss_mask = labels != label_pad_token_id

    # dummy token; we'll ignore the losses on these tokens later
    labels[labels == label_pad_token_id] = 0

    per_token_logps = torch.gather(logits.log_softmax(-1), dim=2, index=labels.unsqueeze(2)).squeeze(2)

    if average_log_prob:
        return (per_token_logps * loss_mask).sum(-1) / loss_mask.sum(-1)
    else:
        return (per_token_logps * loss_mask).sum(-1)
```

Seleccione el log-prob correspondiente al siguiente token

Sume los log-probs

Transformadores para una mejor eficiencia

Reformer:

- La atención del producto punto se reemplaza con la atención de hash sensible a la localidad (LSH)
- Esto logra la atención con $O(nlog(n))$ en lugar del costo de la memoria $O(n^2)$

Kitaev, N., Kaiser, Ł. and Levskaya, A., 2020. Reformer: The efficient transformer. arXiv preprint arXiv:2001.04451. https://arxiv.org/abs/2001.04451

ALBERT:

- Tamaño 5 veces más pequeño que BERT con el mismo rendimiento, debido a la compresión mediante poda

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P. and Soricut, R., 2019. Albert: A lite BERT for self-supervised learning of language representations. arXiv

- preprint arXiv:1909.11942. https://arxiv.org/abs/1909.11942