Práctica 3

Flava

Primero empecé probando con el modelo de FLAVA (Foundational Language–And Vision Alignment) facebook/flava-full, que es un VLM multimodal generativo para aprender representaciones conjuntas de imágenes y texto para múltiples tareas de visión y lenguaje. Esta primera aproximación está en el archivo flava.py. Para mi caso en concreto que tengo que emparejar una imagen con su descripción, no funciona muy bien porque no fue entrenado para esa tarea en específico a diferencia de CLIP. Además, con CLIP los dos embeddings (imagen y texto), están en el mismo espacio vectorial y podía hacer directamente mediante la similitud coseno.

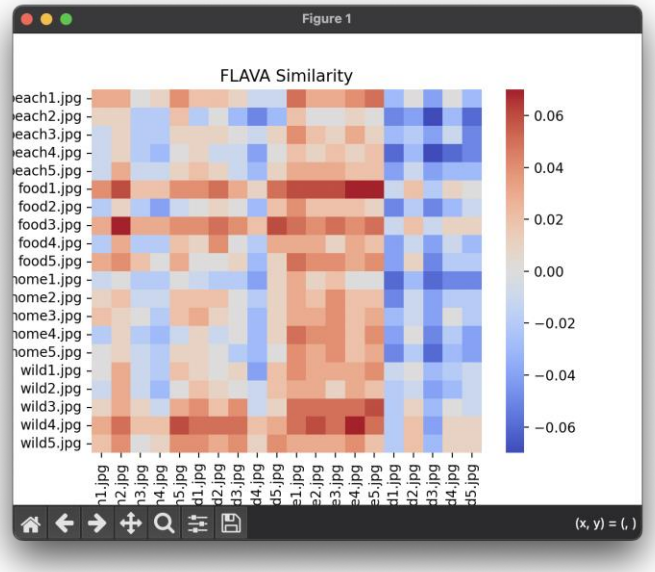
```
# ---------------------------
# SIMILITUD COSENO
# ---------------------------
similarities = image_embeddings @ text_embeddings.T  # (num_images, num_texts)
```

En cambio, FLAVA devuelve embeddings tokenizados, un vector por cada patch (patch es el fragmento de una imagen, no una imagen) visual y otro por cada token (un token textual son las palabras que dependiendo del contexto tienen un significado u otro) textual, en lugar de un único embedding global por imagen o texto. Esta diferencia hizo que al principio tuviera errores mi programa. Para solucionarlos y poder hacer la similitud coseno, necesitaba un único embedding por imagen y uno por texto. Entonces, hice el promedio de todos los patch/tokens, luego los añadí en una lista y concatené los elementos:

```
        # Promediar sobre tokens/patches
        img_emb = F.normalize(outputs.image_embeddings.mean(dim=1), dim=1)
        txt_emb = F.normalize(outputs.text_embeddings.mean(dim=1), dim=1)

    image_embeddings.append(img_emb)
    text_embeddings.append(txt_emb)

image_embeddings = torch.cat(image_embeddings, dim=0)
text_embeddings = torch.cat(text_embeddings, dim=0)
similarities = image_embeddings @ text_embeddings.T
```

Cómo resultado me da:

```
Top-1 accuracy (FLAVA): 10.00%
```



Da una precisión muy baja y en el mapa de calor, las celdas diagonales (pares imagen-caption correctos) no destacan de forma clara frente al resto. Esto es debido a que FLAVA para esta tarea en concreto no es el más adecuado ya que, a diferencia de CLIP que se usa para emparejar una caption ya creada a una imagen ya generada, está diseñado como un modelo multimodal generalista, que busca aprender representaciones conjuntas para múltiples tareas. Además, mi solución para que funcionara con la distancia coseno es bastante rudimentaria.

FLAVA 2

Mi segunda aproximación fue utilizar FLAVA con mi dataset pero para su propósito específico, aprender representaciones conjuntas de imágenes y texto, en vez de intentar ajustarlo a emparejar imágenes y texto con la distancia coseno. Está el código en el archivo flava2.py

En la implementación, primero preprocesé el batch completo:

```
# Preprocesar batch completo
inputs = processor(
    text=captions,
    images=images,
    return_tensors="pt",
    padding="max_length",
    max_length=77  # longitud máxima para todos los textos
)
inputs = {k: v.to(device) for k, v in inputs.items()}
```

Preparé todas las imágenes y captions en un solo batch. padding="max_length" asegura que todos los textos tengan la misma longitud (max_length=77).

Luego convierto todo a tensores PyTorch y los movería a la GPU si estuviera disponible, pero en mi caso no porque tengo un mac y procesa todo en la CPU.

```
# Obtener embeddings
with torch.no_grad():
    outputs = model(**inputs)
    # Colapsar la dimensión de tokens/patches para obtener vectores de tamaño fijo
    image_embeddings = F.normalize(outputs.image_embeddings.mean(dim=1), dim=-1)       # [
    text_embeddings = F.normalize(outputs.text_embeddings.mean(dim=1), dim=-1)         # [
    multimodal_embeddings = F.normalize(outputs.multimodal_embeddings.mean(dim=1), dim=-1)
```

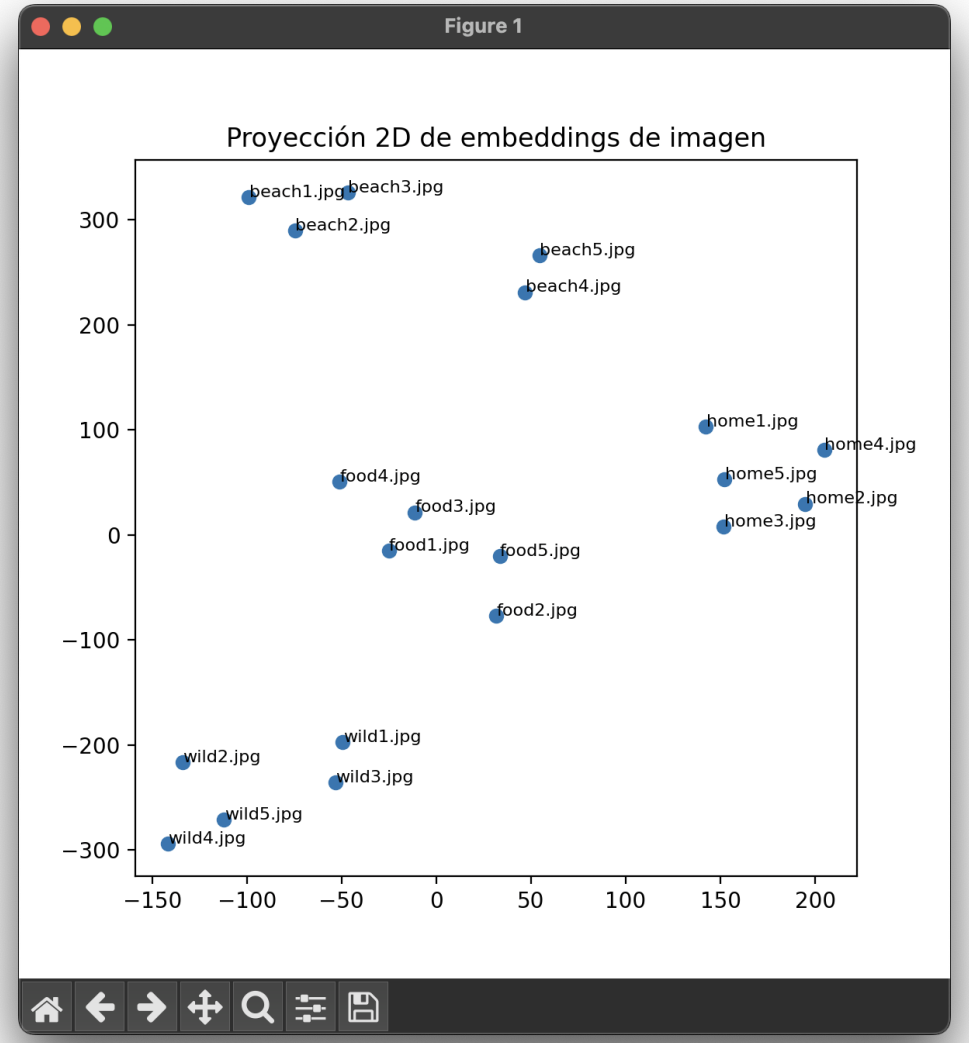
Más tarde, como siempre hago, dentro del with torch.no_grad(): (que desactiva el cálculo de gradientes para ahorrar memoria), recibir los outputs y almacenarlos en una variable y la normalización y promedio de los embeddings por separado de imagen, texto y multimodales.

```
Image embeddings shape: torch.Size([20, 768])
Text embeddings shape: torch.Size([20, 768])
Multimodal embeddings shape: torch.Size([20, 768])
```

Después compruebo la forma que tienen los embeddings para ver si está correcto y tienen las mismas dimensiones para su posterior análisis.

```
tsne = TSNE(n_components=2, perplexity=5, random_state=42)
image_2d = tsne.fit_transform(image_embeddings.numpy())

plt.figure(figsize=(6,6))
plt.scatter(image_2d[:,0], image_2d[:,1])
for i, fname in enumerate(df["filename"]):
    plt.text(image_2d[i,0]+0.1, image_2d[i,1]+0.1, fname, fontsize=8)
plt.title("Proyección 2D de embeddings de imagen")
plt.show()
```

Por último, hice la visualización de los resultados con TSNE. Primero reduce las dimensiones de los embeddings pasando de 768 a dos para su visualización. Puse una perplexity de 5 ya que mi dataset tiene 5 imágenes por categoría (la perplexity indica cuántos vecinos cercanos “cree” t-SNE que tiene cada punto). Luego ya realiza una gráfica de puntos en el espacio de 2 dimensiones.



Podemos observar cómo el resultado es muy bueno con 5 grupos bien diferenciados y coinciden todos con la categoría que se les había asignado previamente. Hay una diferencia entre tres playas (beach1, beach2 y beach3) y las otras 2 ya que las primeras 3 son playas muy genéricas y de las primeras que aparecen en Google y las otras 2 son playas de A Coruña.

De esta manera se puede concluir que FLAVA está funcionando perfectamente y está entendiendo y sabiendo diferenciar las 5 categorías, tanto el texto como la imagen.

BLIP

Luego hice BLIP (Bootstrapped Language-Image Pretraining), que mejora su propio entrenamiento combinando: datos reales (pares imagen-texto), datos generados por el propio modelo (pseudo-captions) y filtros de calidad que seleccionan las mejores

descripciones. Además, a diferencia de CLIP, fue diseñado para entender, generar y razonar sobre imagen+texto, no solo mirar la similitud coseno de pares de imagen-texto.

Comparado con CLIP también tuve que hacer unos cuántos cambios en el código para que funcionase.
Primero, para recibir los outputs del modelo dependiendo de las entradas en CLIP ya se podía hacer primero la variable outputs y luego separar en salidas de imágenes y salidas de texto:

```
outputs = model(**inputs)
```

Y ya daba directamente los embeddings de las imágenes y las de los textos. En cambio, con mi modelo de BLIP "Salesforce/blip-itm-base-coco", devuelve un BlipImageTextMatchingModelOutput, que solo tiene los logits de matching, no los embeddings internos. Entonces, accedí directamente a los embeddings de cada uno sin pasar por otra variable previa:

```
# Extraer features visuales del encoder
vision_outputs = model.vision_model(pixel_values=inputs["pixel_values"])
image_feat = vision_outputs.pooler_output  # [CLS] del encoder visual
image_emb = model.vision_proj(image_feat)  # proyectado al espacio multimodal

# Extraer features textuales del encoder
text_outputs = model.text_encoder(
    input_ids=inputs["input_ids"],
    attention_mask=inputs["attention_mask"],
    return_dict=True
)
```

Además, en el encoder visual, a diferencia de CLIP, el ViT (vision transformer) produce una secuencia de patch embeddings, pero la capa pooler_output da el embedding del token [CLS] visual. Ese embedding no está en el espacio multimodal aún, es decir, en el mismo espacio vectorial que los embeddings de texto. Es solo una representación visual "cruda" del ViT.
Por eso hice:

```
image_emb = model.vision_proj(image_feat)
```

Que es la capa lineal que tiene BLIP llamada vision_proj que transforma la feature visual al espacio conjunto visión-texto. Y ya estarían alineados los embeddings de imagen y texto con la misma dimensionalidad.

Para el encoder de texto:

```
# Extraer features textuales del encoder
text_outputs = model.text_encoder(
    input_ids=inputs["input_ids"],
    attention_mask=inputs["attention_mask"],
    return_dict=True
)
text_feat = text_outputs.last_hidden_state[:, 0, :]  # [CLS]
```

Primero se pasa el caption por un BERT base. Después se toma last_hidden_state[:, 0, :], que es el token [CLS] textual, igual que en BERT normal.

```
text_emb = model.text_proj(text_feat)
```

Y al igual que con el encoder visual, obtengo la capa lineal que se llama text_proj para que los embeddings de texto e imagen estén en el mismo espacio para calcular la similitud.

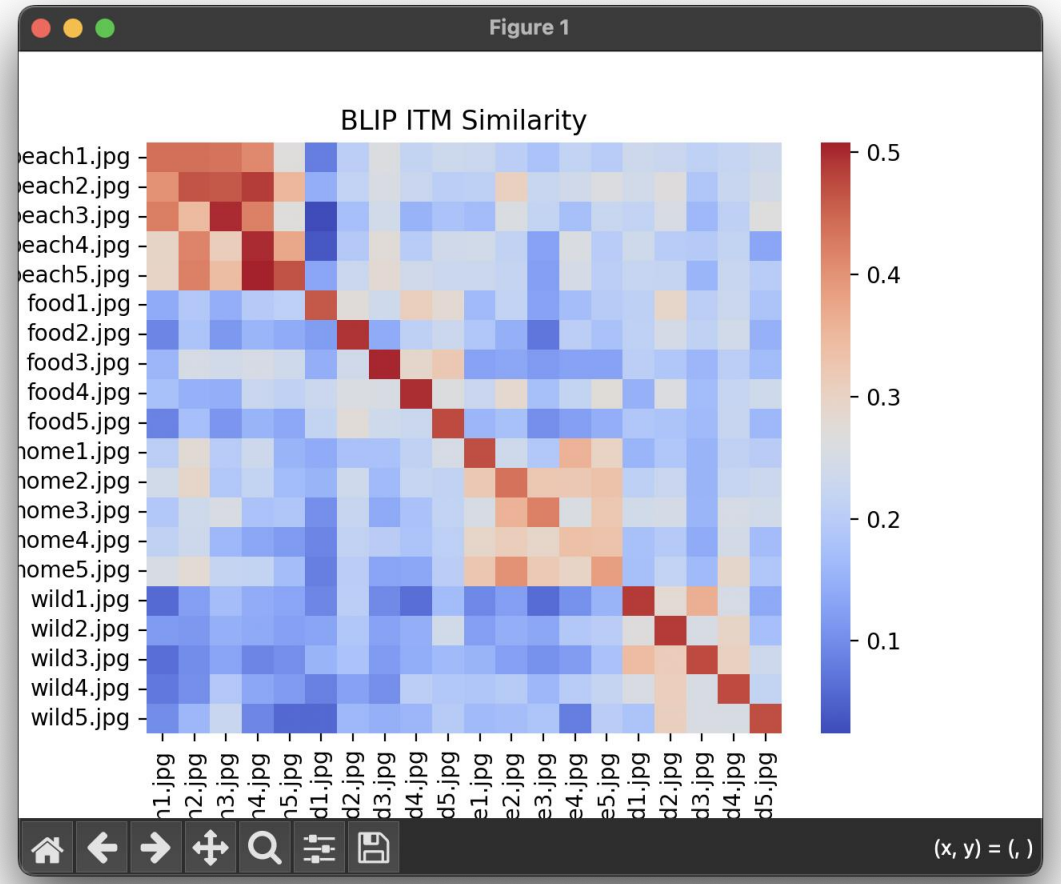
Luego normalizé los embeddings al igual que FLAVA, primero cada embedding lo normalicé y luego los concatené.

```
        # Normalizar
        img_emb = F.normalize(image_emb, dim=-1)
        txt_emb = F.normalize(text_emb, dim=-1)

    image_embs.append(img_emb)
    text_embs.append(txt_emb)

# Similaridad global
image_embs = torch.cat(image_embs, dim=0)
text_embs = torch.cat(text_embs, dim=0)
```

Hice dim=-1 porque los embeddings tienen la forma (batch_size, embedding_dim), batch_size es la cantidad de embeddings y embedding_dim es la dimensión de cada embedding (256 por ejemplo) y cuando pongo –1 pytorch lo traduce a 1 porque la última dimensión (embedding_dim) es la 1.

Esto me da unos resultados muy buenos:

```
Top-1 accuracy (BLIP ITM embeddings): 80.00%
```



80% de top-1 accuracy está muy bien y más sabiendo que no simplemente empareja imagen y texto si no que fue diseñado para entender, generar y razonar sobre imagen+texto.

En mi heatmap se puede ver como en la categoría beach es claramente más rojo que el resto de las categorías, esto indica que BLIP identifica muy bien que las descripciones del grupo beach coinciden con imágenes beach y las imágenes entre ellas también son muy similares en embeddings. Esto puede ser porque en las caption incluí en todas la palabra playa. Además, las imágenes son más parecidas entre ellas comparado con las imágenes de otras categorías. El segundo grupo más rojo es el de home pienso.

Luego en blip2.py generé las caption generada de cada una de las imágenes:

beach1.jpg → a beach with palm trees and a white sand beach

beach2.jpg → a beach with a few palm trees and a blue sky

beach3.jpg → a beach with palm trees and the ocean

beach4.jpg → a body of water

beach5.jpg → a beach with a body of water and buildings

food1.jpg → a plate of food with a bowl of salsa and a bowl of salsa

food2.jpg → a hamburger with cheese, let and let

food3.jpg → pain pain de pain de pain de pain de pain de pain de pain de pain de pain

food4.jpg → a pizza with a slice missing

food5.jpg → a bowl of vegetables and a salad on a wooden table

home1.jpg → a kitchen with a dining table and chairs

home2.jpg → a living room with a couch, coffee table and television

home3.jpg → a living room with a fireplace and a couch

home4.jpg → a living room with a couch and a staircase

home5.jpg → a living room with a staircase leading to a dining area

wild1.jpg → a lion laying on the ground

wild2.jpg → an elephant standing in a field of grass

wild3.jpg → a leopard sitting in the grass under a tree

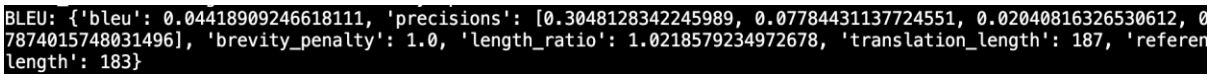
wild4.jpg → a zebra grazing in a field

wild5.jpg → three gis walking in a field with trees in the background

Me dan resultados bastante buenos como: “a beach with palm trees and a white sand beach”, “a living room with a couch, coffee table and television” y “a leopard sitting in the grass under a tree”. Pero en algunos el modelo alucina un poco: “a plate of food with a bowl of salsa and a bowl of salsa”, “pain pain de pain de pain…”; o muy genéricos: “a body of water”.

El modelo es muy útil para describir imágenes en categorías claras y objetos principales. Pero hay que tener en cuenta que puede generar detalles inventados o repetidos y captions genéricas.

Utilicé cómo métricas cuantitativas para la evaluación del modelo BLEU, mide coincidencia n-grama entre caption generada y referencia. Es muy estricta y penaliza diferencias superficiales en la redacción; y METEOR, que tiene en cuenta sinónimos, lematización y coincidencia semántica, por lo que es más adecuada para evaluar modelos generativos modernos.



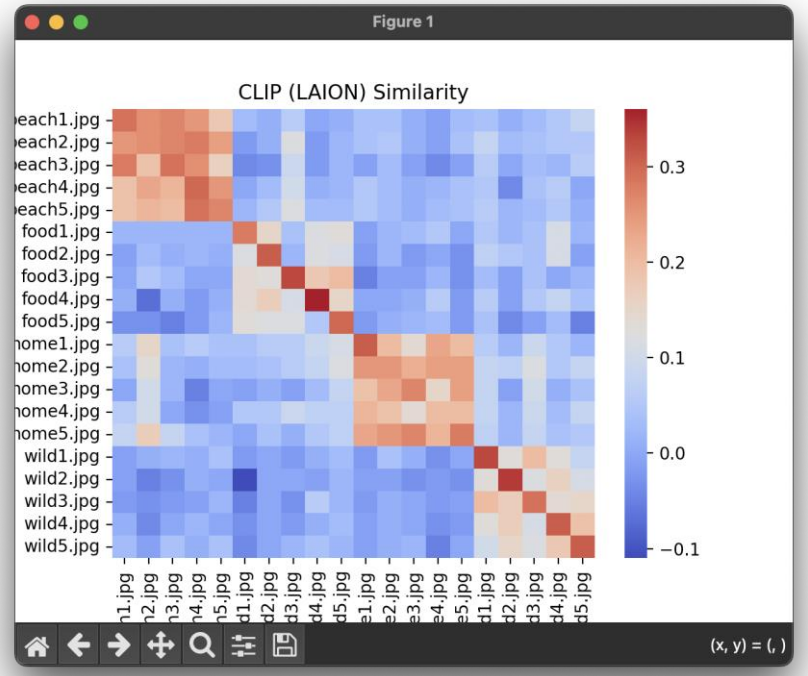
BLEU me da 0.0437, lo cual es muy bajo debido a la falta de coincidencia literal, ya que penaliza mucho que la frase generada no coincida palabra a palabra con la referencia.

Meteor me da 0.286, que es bastante razonable para un dataset pequeño sin fine-tuning. Meteor representa mucho más el rendimiento real ya que considera sinónimos, stemming y coincidencia semántica, por lo que refleja mejor la calidad de la caption generada.

CLIP laion

CLIP Laion es una implementación abierta de CLIP entrenada sobre el dataset LAION-400M/5B, que es:

- LAION-400M → ~400 millones de pares imagen-texto.
- LAION-5B → versión más grande con ~5.8 mil millones de pares.
- Dataset no filtrado profesionalmente, pero enorme y diverso.
- Entrenamiento contrastivo estilo CLIP sobre estas imágenes y sus captions.



Muy buen rendimiento para emparejar captions e imágenes con un 85%. Además, la matriz de correlaciones muestra fuerte relación entre elementos de la misma categoría, destacando beach y home.

Conclusión

¿Qué modelo VLM se ha escogido y por qué?

Dependiendo del propósito, elegiría un modelo u otro. CLIP para simplemente emparejar captions ya creadas con imágenes, BLIP para generar captions en base a imágenes y FLAVA para generar embeddings que describen una imagen y/o un texto, pero no genera una descripción de la imagen. Para este caso en concreto me parece más interesante BLIP ya que, en el caso de utilizar el modelo con una imagen que no ha sido insertada en el dataset previamente y aún no tiene una caption que la describa, podría dar una solución factible. En cambio, CLIP solo la emparejaría con una descripción de una imagen que sea parecida. Además, BLIP me dio un 80% de top-1 accuracy, lo cual está muy bien, y más sabiendo que son descripciones generadas, no emparejadas.

¿El modelo escogido responde como es esperado al prompt y a las imágenes?

Probando con las imágenes del dataset, BLIP da resultados bastante buenos, pero también tiene alucinaciones o hace captions muy genéricas.

¿Qué métricas se usaron para la evaluación cuantitativa y por qué?

Para la evaluación cuántitativa primero utilizé BLEU porque es una primera aproximación muy estricta y poco representativa sobretodo para compararlo luego con el rendimiento de la otra métrica de evaluación cuantitativa. Además, porque es más estricta y si coincide palabra a palabra. Después utilicé METEOR, que representa mucho mejor el rendimiento real, ya que es más fiable, indica que el modelo captura la semántica correcta y la categoría principal de las imágenes.

¿Qué clases o casos funcionaron mejor y cuáles peor?

Funciona mejor con wild y home, que hace todas las descripciones bastante bien. Con las imágenes de la categoría beach hace buenas descripciones, pero hay una que es bastante genérica: “a body of water”. Food es la que tiene peor rendimiento con varias alucinaciones: “a plate of food with a bowl of salsa and a bowl of salsa”, “pain pain de pain de pain…” y “a hamburger with cheese, let and let”.

¿Son las métricas escogidas representativas del rendimiento del modelo?

BLEU en este caso no es representativa, pero está bien para saber si hubiera hecho un fine-tuning, para comprobar si está realizando las descripciones iguales a las integradas en el entrenamiento. Por otro lado, METEOR sí que es más representativa, captura semántica y objetos principales.

¿Qué limitaciones se han encontrado en el modelo o en el dataset?

Las limitaciones que he encontrado son que el dataset es muy pequeño, las imágenes de food quizás eran muy complejas o ambiguas y por eso BLIP alucina y que sin fine-tuning, BLIP no se adapta a peculiaridades del dataset.

¿El tiempo de inferencia afecta a la usabilidad del modelo en un caso real? ¿Podría usarse en tiempo real? ¿Se requeriría hardware especializado?

El tiempo de inferencia de BLIP en un caso real es bajo con GPU y podría usarse en tiempo real para aplicaciones sencillas sin un hardware especializado extremo.