OpenVINO™
CVPR 2025

CVPR Nashville JUNE 11-15, 2025

Cognitive AI for the Future: Multimodal Models and RAG in Vision Language Applications, from Training to Deployment

Module 2: Finetuning Embedding Models and LVLM

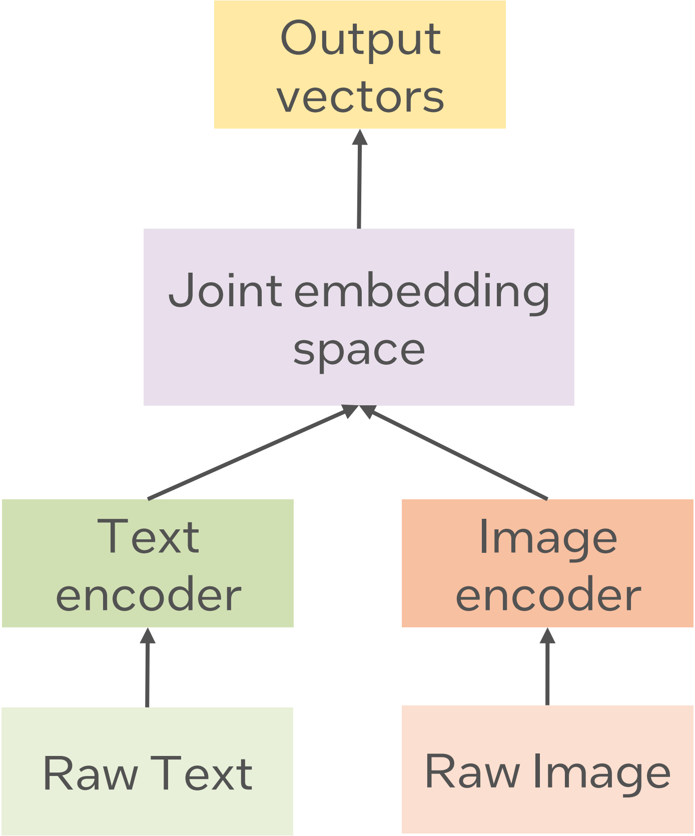
Speaker: Gustavo Lujan and Tiep Le
Job title: AI Research Scientist

What is Cognitive AI in the real-world ?

- **E-commerce Image-Text Search**: Bridges product photos and text queries (e.g., “red sneakers with white soles”) into one vector space.
- **Visual Question Answering for Support**: Embeds a user’s photo of a broken device and their question jointly, retrieving precise answers from a knowledge base.
- **Medical Image–Report Retrieval**: Maps new radiology scans and past diagnostic reports into the same space to surface similar historical cases.
- **Multimodal Video Segment Search**: Indexes video frames and transcribed dialogue together so queries like “scenes discussing climate change” return exactly those segments

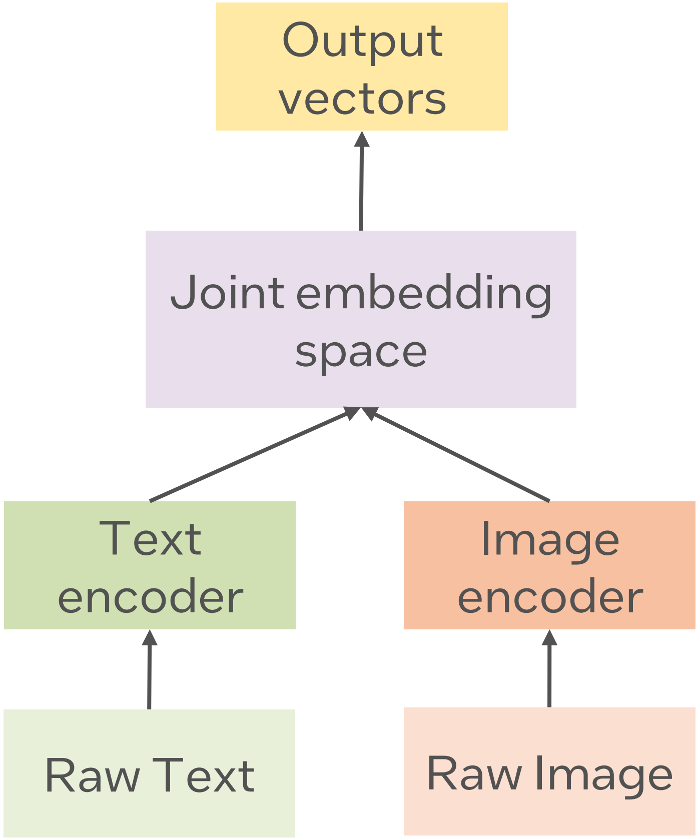
Why an Embedding Model?

- Captures Meaning: Transforms raw text or image data into vectors that reflect semantic content.
- Enables Similarity Search: Positions related items close in vector space, allowing efficient retrieval of comparable texts or visually similar images.
- Supports Multimodal Alignment: When trained jointly, text and image embeddings reside in a shared space.



Why an Embedding Model?

- **Reduces Complexity**: Converts high-dimensional inputs into manageable, fixed-size vectors, making storage and computation more efficient.
- **Drives Scalable Indexing**: Vector indexes (e.g., FAISS) operate on embeddings to provide fast, sublinear search across large datasets.
- **Facilitates Unified Pipelines**: Treats all data as vectors, so downstream components don't need separate logic for text versus images—everything is a vector lookup.


Output vectors

Joint embedding space

Text encoder

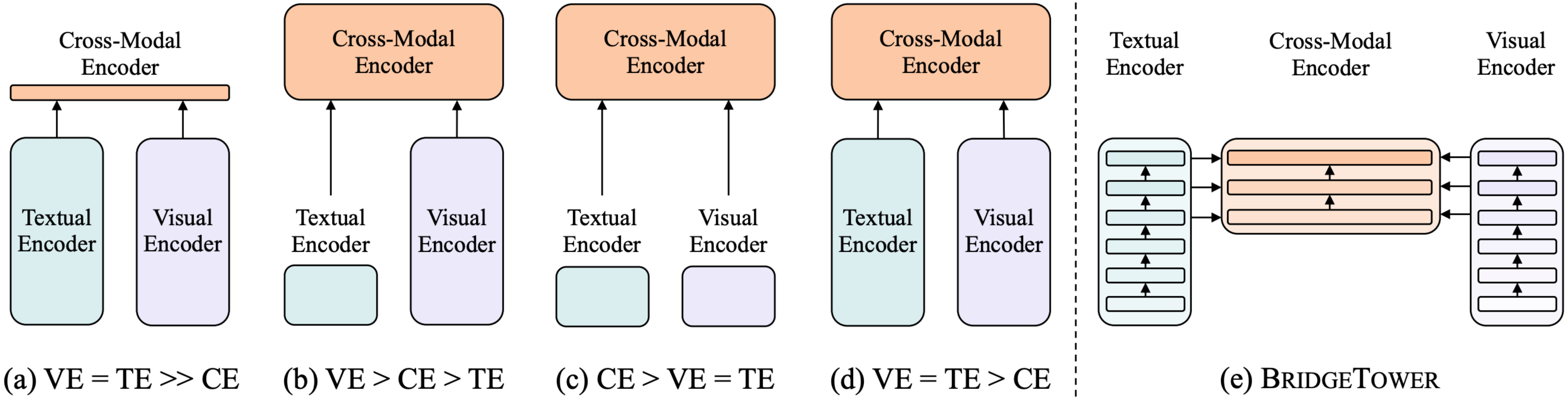
Image encoder

Raw Text

Raw Image

What is BridgeTower?

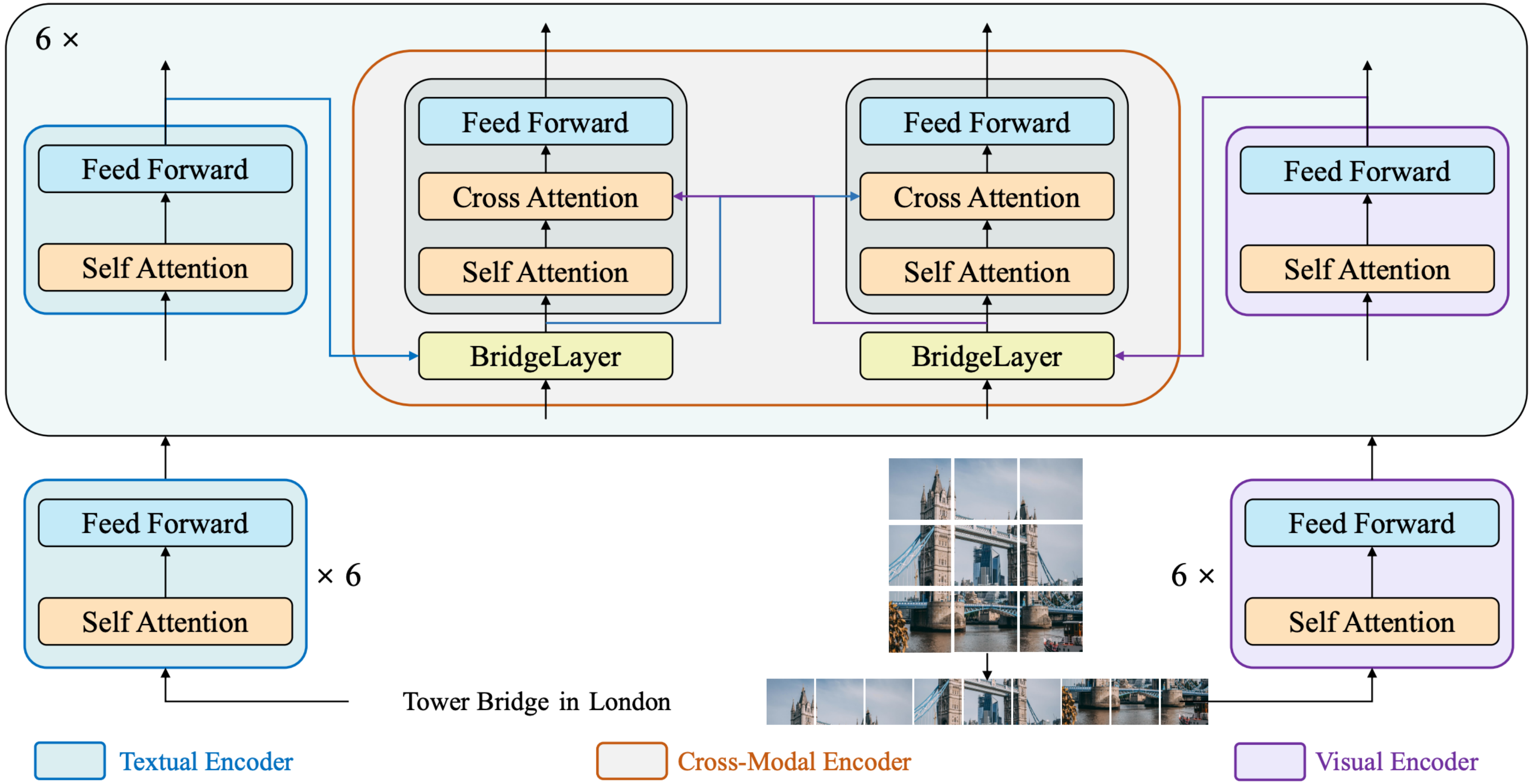
Building Bridges Between Encoders in Vision-Language Representation Learning



Four categories of two-tower vision-language models are shown in panels (a)–(d).
Panel (e) illustrates the BRIDGETOWER architecture, where:

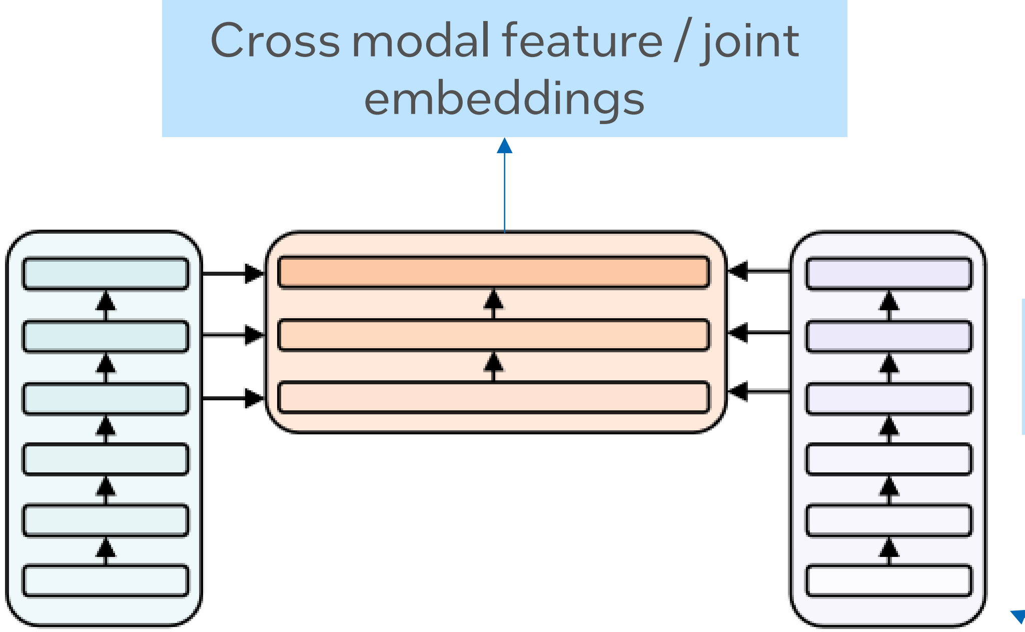
- Each block's height reflects its relative computational cost
- The Visual Encoder (VE) and Textual Encoder (TE) have similar size/complexity
- A Cross-modal Encoder (CE) fuses visual and textual features

BridgeTower Architecture Overview



- Separate 12-layer text encoder and 12-layer visual encoder
- 6-layer cross-modal encoder for joint processing
- Bridge connections from the top 6 layers of both text and visual encoders into each cross-modal layer

Joint embeddings between text and images

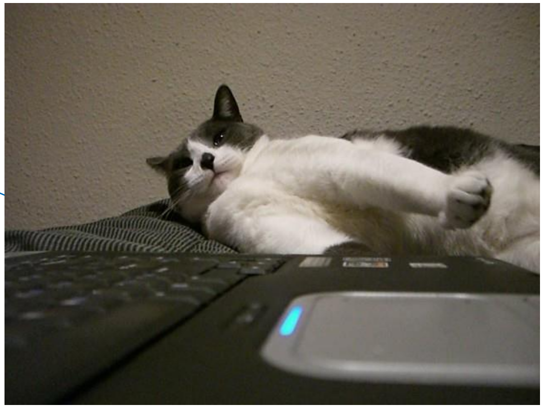


Cross modal feature / joint embeddings

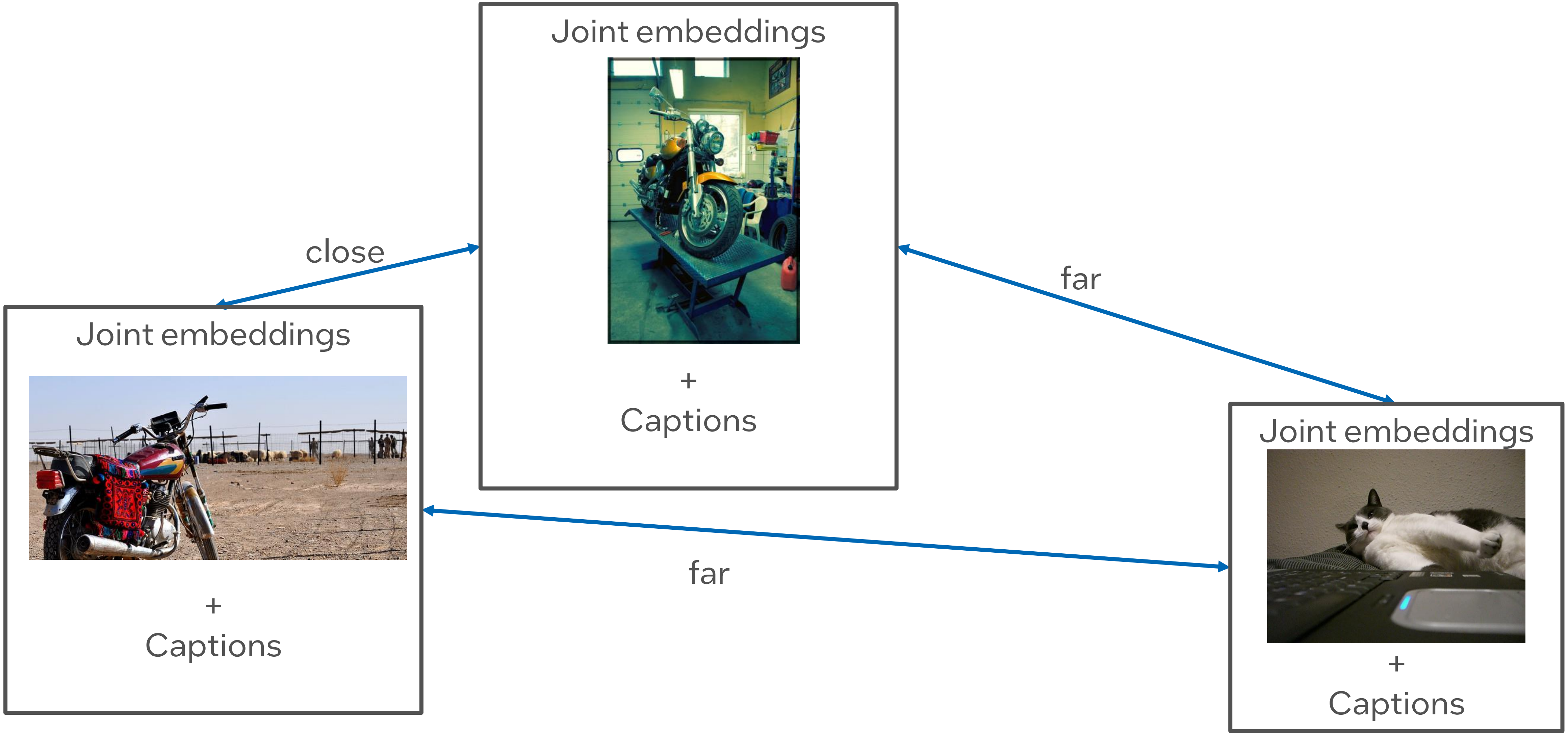
Text encoder

Image encoder

Audio transcript: a cat laying down stretched out near a laptop

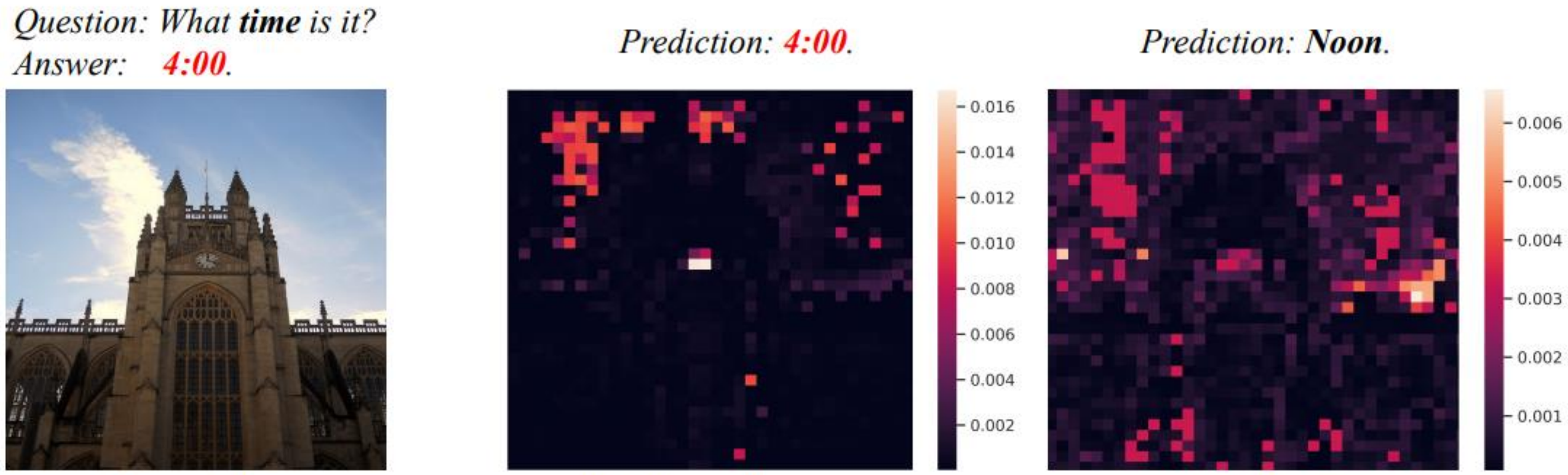


Distance in Embedding Space between Pairs of images/video (frames) + captions (UMAP)



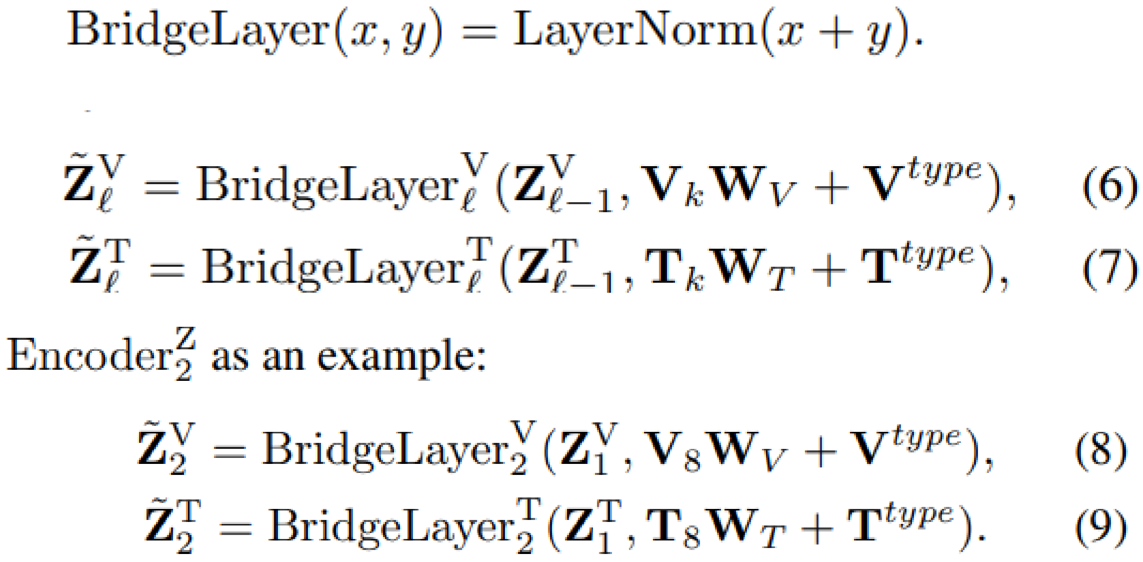
BridgeTower main advantages

- BridgeTower adds bridge layers from the top of each uni-modal encoder into every layer of the cross-modal encoder, allowing multi-level semantic features to be fused rather than relying only on last-layer outputs.
- This multi-level fusion comes with almost no extra parameters or compute cost, enabling richer cross-modal alignment without significant overhead.



(a) Original Image (b) BRIDGETOWER (c) METER

BridgeTower



$$\mathrm{BridgeLayer}(x, y) = \mathrm{LayerNorm}(x + y).$$

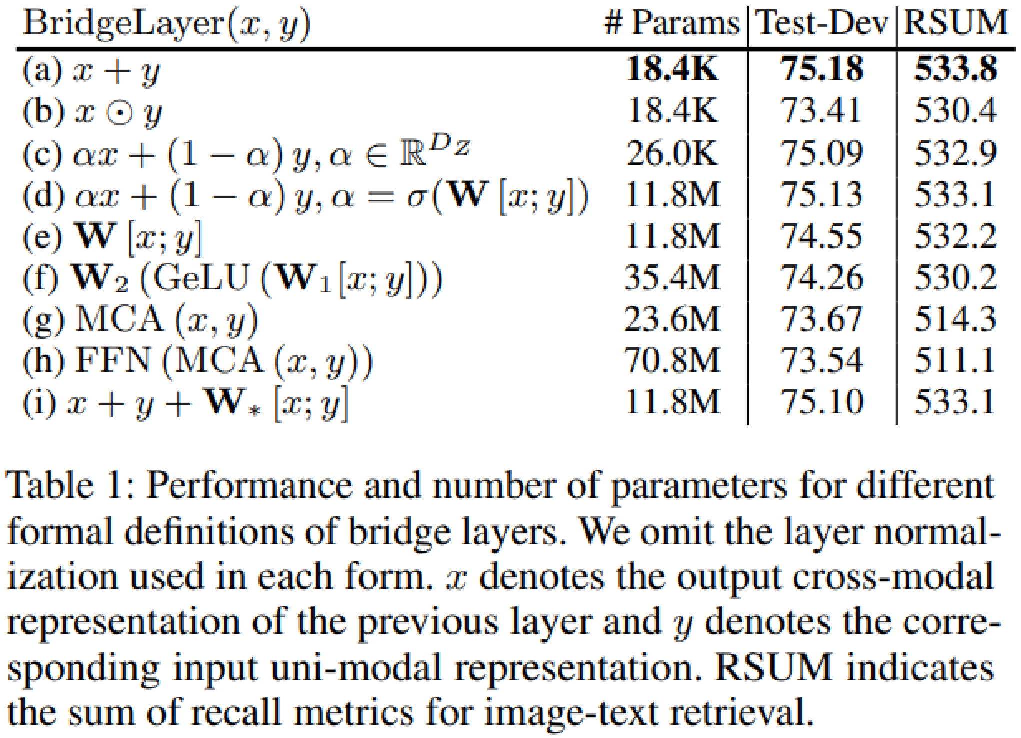
$$\tilde{\mathbf{Z}}_\ell^{\mathrm{V}} = \mathrm{BridgeLayer}_\ell^{\mathrm{V}}(\mathbf{Z}_{\ell-1}^{\mathrm{V}}, \mathbf{V}_k\mathbf{W}_V + \mathbf{V}^{type}), \quad (6)$$

$$\tilde{\mathbf{Z}}_\ell^{\mathrm{T}} = \mathrm{BridgeLayer}_\ell^{\mathrm{T}}(\mathbf{Z}_{\ell-1}^{\mathrm{T}}, \mathbf{T}_k\mathbf{W}_T + \mathbf{T}^{type}), \quad (7)$$

$\mathrm{Encoder}_2^{\mathrm{Z}}$ as an example:

$$\tilde{\mathbf{Z}}_2^{\mathrm{V}} = \mathrm{BridgeLayer}_2^{\mathrm{V}}(\mathbf{Z}_1^{\mathrm{V}}, \mathbf{V}_8\mathbf{W}_V + \mathbf{V}^{type}), \quad (8)$$

$$\tilde{\mathbf{Z}}_2^{\mathrm{T}} = \mathrm{BridgeLayer}_2^{\mathrm{T}}(\mathbf{Z}_1^{\mathrm{T}}, \mathbf{T}_8\mathbf{W}_T + \mathbf{T}^{type}). \quad (9)$$

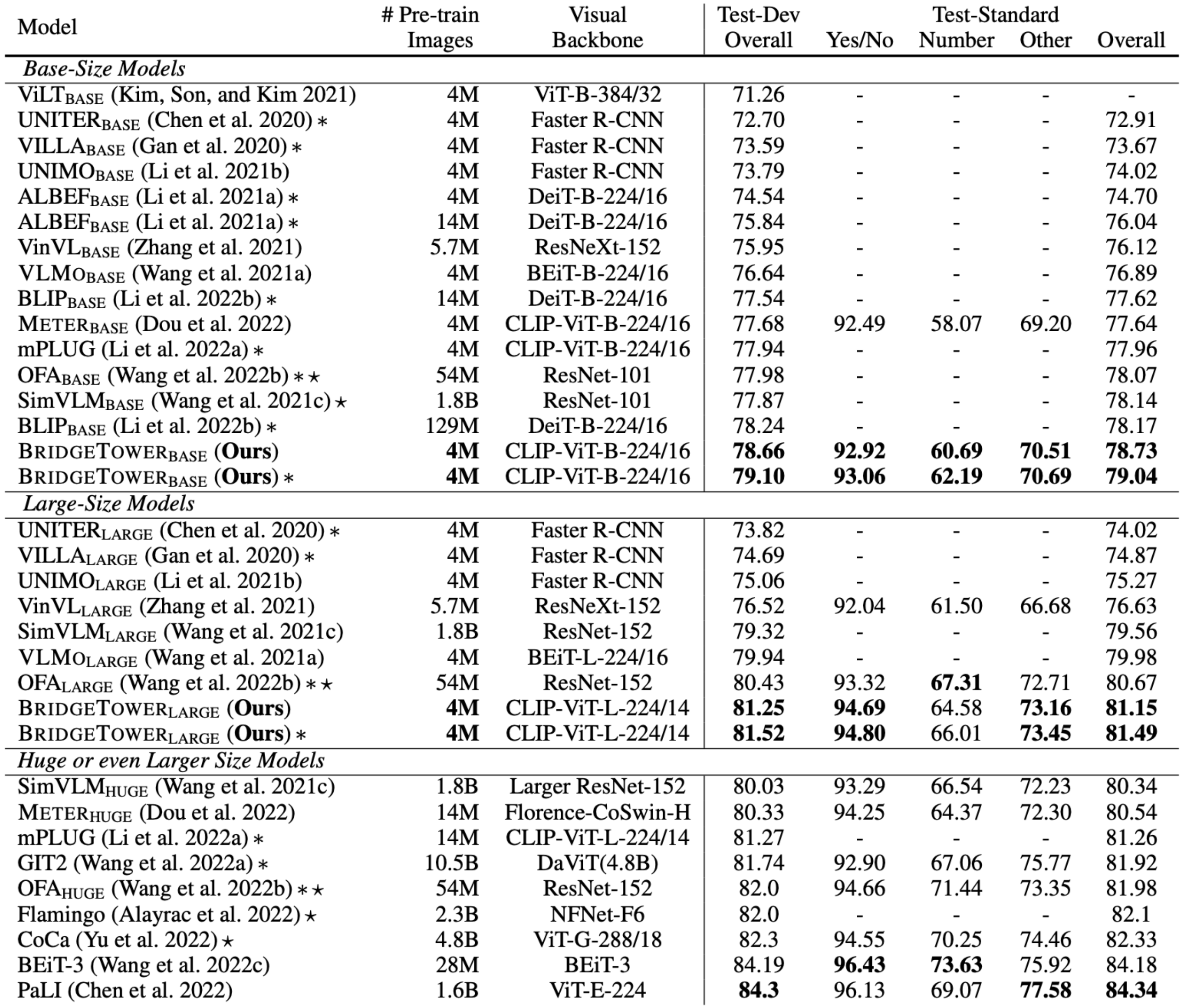


BridgeLayer(x, y)	# Params	Test-Dev	RSUM
(a) $x + y$	**18.4K**	**75.18**	**533.8**
(b) $x \odot y$	18.4K	73.41	530.4
(c) $\alpha x + (1 - \alpha)\, y, \alpha \in \mathbb{R}^{Dz}$	26.0K	75.09	532.9
(d) $\alpha x + (1 - \alpha)\, y, \alpha = \sigma(\mathbf{W}\,[x; y])$	11.8M	75.13	533.1
(e) $\mathbf{W}\,[x; y]$	11.8M	74.55	532.2
(f) $\mathbf{W}_2\,(\mathrm{GeLU}\,(\mathbf{W}_1[x; y]))$	35.4M	74.26	530.2
(g) $\mathrm{MCA}\,(x, y)$	23.6M	73.67	514.3
(h) $\mathrm{FFN}\,(\mathrm{MCA}\,(x, y))$	70.8M	73.54	511.1
(i) $x + y + \mathbf{W}_*\,[x; y]$	11.8M	75.10	533.1

Table 1: Performance and number of parameters for different formal definitions of bridge layers. We omit the layer normalization used in each form. x denotes the output cross-modal representation of the previous layer and y denotes the corresponding input uni-modal representation. RSUM indicates the sum of recall metrics for image-text retrieval.

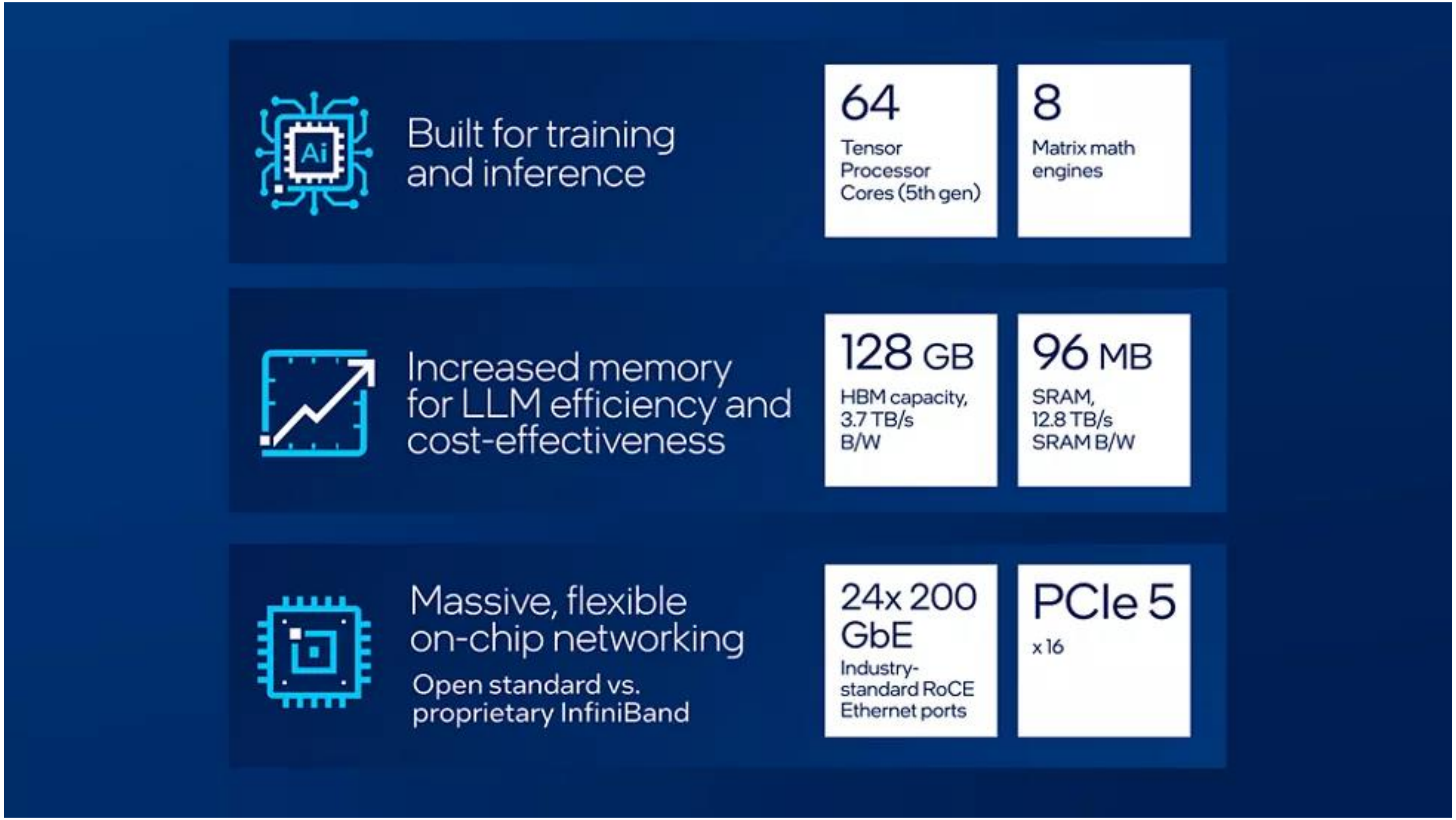
BridgeTower Results

- BridgeTowerBase, with just 4 M pairs, outperforms two-tower models that use 10×–1 000× more data.
- BridgeTowerLarge scores 81.15 % on VQAv2 test-std.
- Both base and large BridgeTower models surpass comparable two-tower approaches, showing multi-level fusion's clear advantage.



Model	# Pre-train Images	Visual Backbone	Test-Dev Overall	Test-Standard Yes/No	Number	Other	Overall
Base-Size Models							
ViLT$_{\text{BASE}}$ (Kim, Son, and Kim 2021)	4M	ViT-B-384/32	71.26	-	-	-	-
UNITER$_{\text{BASE}}$ (Chen et al. 2020) ∗	4M	Faster R-CNN	72.70	-	-	-	72.91
VILLA$_{\text{BASE}}$ (Gan et al. 2020) ∗	4M	Faster R-CNN	73.59	-	-	-	73.67
UNIMO$_{\text{BASE}}$ (Li et al. 2021b)	4M	Faster R-CNN	73.79	-	-	-	74.02
ALBEF$_{\text{BASE}}$ (Li et al. 2021a) ∗	4M	DeiT-B-224/16	74.54	-	-	-	74.70
ALBEF$_{\text{BASE}}$ (Li et al. 2021a) ∗	14M	DeiT-B-224/16	75.84	-	-	-	76.04
VinVL$_{\text{BASE}}$ (Zhang et al. 2021)	5.7M	ResNeXt-152	75.95	-	-	-	76.12
VLMo$_{\text{BASE}}$ (Wang et al. 2021a)	4M	BEiT-B-224/16	76.64	-	-	-	76.89
BLIP$_{\text{BASE}}$ (Li et al. 2022b) ∗	14M	DeiT-B-224/16	77.54	-	-	-	77.62
METER$_{\text{BASE}}$ (Dou et al. 2022)	4M	CLIP-ViT-B-224/16	77.68	92.49	58.07	69.20	77.64
mPLUG (Li et al. 2022a) ∗	4M	CLIP-ViT-B-224/16	77.94	-	-	-	77.96
OFA$_{\text{BASE}}$ (Wang et al. 2022b) ∗⋆	54M	ResNet-101	77.98	-	-	-	78.07
SimVLM$_{\text{BASE}}$ (Wang et al. 2021c) ⋆	1.8B	ResNet-101	77.87	-	-	-	78.14
BLIP$_{\text{BASE}}$ (Li et al. 2022b) ∗	129M	DeiT-B-224/16	78.24	-	-	-	78.17
BRIDGETOWER$_{\text{BASE}}$ **(Ours)**	**4M**	CLIP-ViT-B-224/16	**78.66**	**92.92**	**60.69**	**70.51**	**78.73**
BRIDGETOWER$_{\text{BASE}}$ **(Ours)** ∗	**4M**	CLIP-ViT-B-224/16	**79.10**	**93.06**	**62.19**	**70.69**	**79.04**
Large-Size Models							
UNITER$_{\text{LARGE}}$ (Chen et al. 2020) ∗	4M	Faster R-CNN	73.82	-	-	-	74.02
VILLA$_{\text{LARGE}}$ (Gan et al. 2020) ∗	4M	Faster R-CNN	74.69	-	-	-	74.87
UNIMO$_{\text{LARGE}}$ (Li et al. 2021b)	4M	Faster R-CNN	75.06	-	-	-	75.27
VinVL$_{\text{LARGE}}$ (Zhang et al. 2021)	5.7M	ResNeXt-152	76.52	92.04	61.50	66.68	76.63
SimVLM$_{\text{LARGE}}$ (Wang et al. 2021c)	1.8B	ResNet-152	79.32	-	-	-	79.56
VLMo$_{\text{LARGE}}$ (Wang et al. 2021a)	4M	BEiT-L-224/16	79.94	-	-	-	79.98
OFA$_{\text{LARGE}}$ (Wang et al. 2022b) ∗⋆	54M	ResNet-152	80.43	93.32	**67.31**	72.71	80.67
BRIDGETOWER$_{\text{LARGE}}$ **(Ours)**	**4M**	CLIP-ViT-L-224/14	**81.25**	**94.69**	64.58	**73.16**	**81.15**
BRIDGETOWER$_{\text{LARGE}}$ **(Ours)** ∗	**4M**	CLIP-ViT-L-224/14	**81.52**	**94.80**	66.01	**73.45**	**81.49**
Huge or even Larger Size Models							
SimVLM$_{\text{HUGE}}$ (Wang et al. 2021c)	1.8B	Larger ResNet-152	80.03	93.29	66.54	72.23	80.34
METER$_{\text{HUGE}}$ (Dou et al. 2022)	14M	Florence-CoSwin-H	80.33	94.25	64.37	72.30	80.54
mPLUG (Li et al. 2022a) ∗	14M	CLIP-ViT-L-224/14	81.27	-	-	-	81.26
GIT2 (Wang et al. 2022a) ∗	10.5B	DaViT(4.8B)	81.74	92.90	67.06	75.77	81.92
OFA$_{\text{HUGE}}$ (Wang et al. 2022b) ∗⋆	54M	ResNet-152	82.0	94.66	71.44	73.35	81.98
Flamingo (Alayrac et al. 2022) ⋆	2.3B	NFNet-F6	82.0	-	-	-	82.1
CoCa (Yu et al. 2022) ⋆	4.8B	ViT-G-288/18	82.3	94.55	70.25	74.46	82.33
BEiT-3 (Wang et al. 2022c)	28M	BEiT-3	84.19	**96.43**	**73.63**	75.92	84.18
PaLI (Chen et al. 2022)	1.6B	ViT-E-224	**84.3**	96.13	69.07	**77.58**	**84.34**

Intel Gaudi 3 Setup and Specifications



Built for training and inference

64 Tensor Processor Cores (5th gen)	**8** Matrix math engines

Increased memory for LLM efficiency and cost-effectiveness

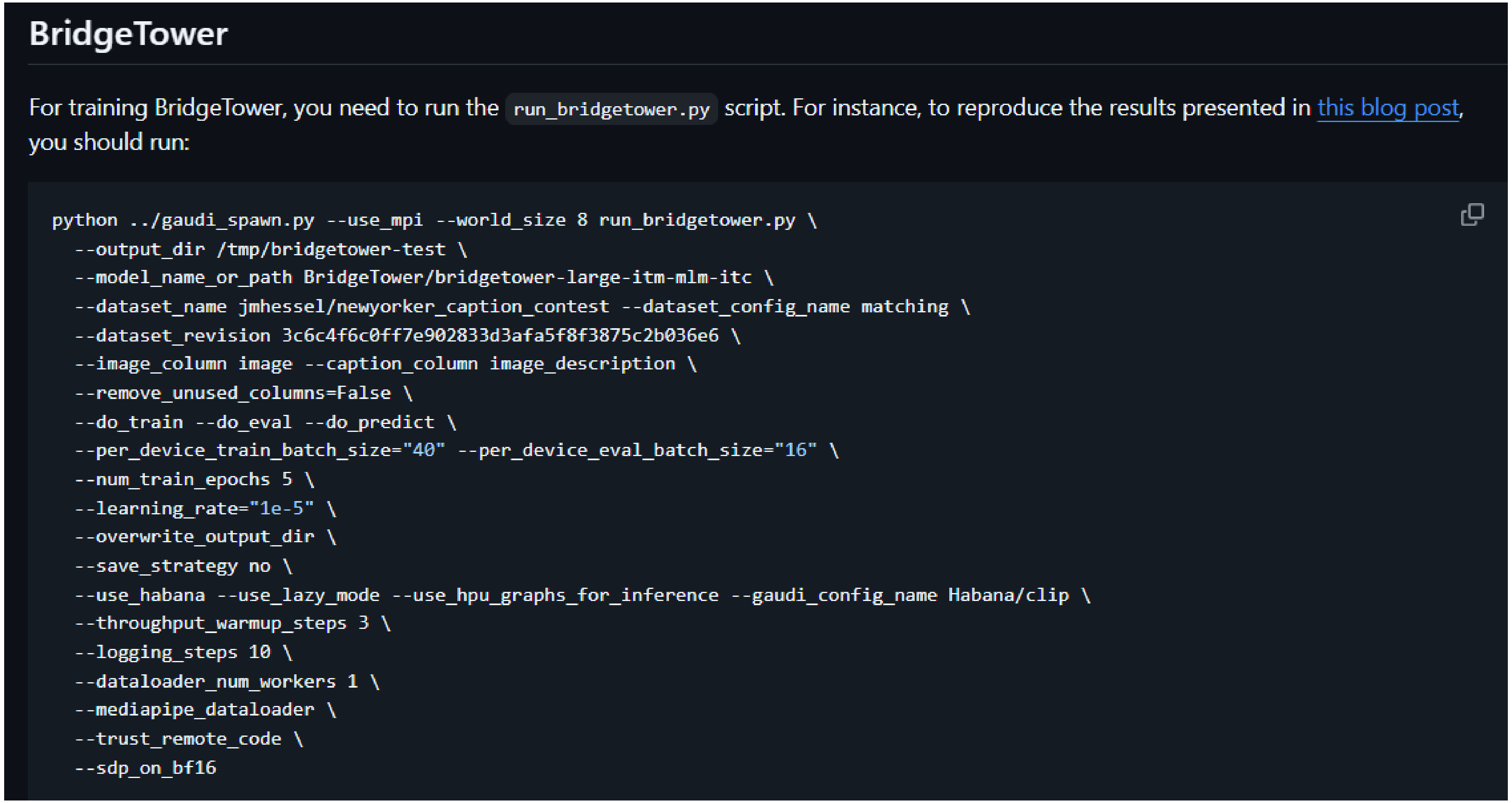
128 GB HBM capacity, 3.7 TB/s B/W	**96 MB** SRAM, 12.8 TB/s SRAM B/W

Massive, flexible on-chip networking

Open standard vs. proprietary InfiniBand

24x 200 GbE Industry-standard RoCE Ethernet ports	**PCIe 5** x 16

BridgeTower Fine-tuning on Gaudi



BridgeTower

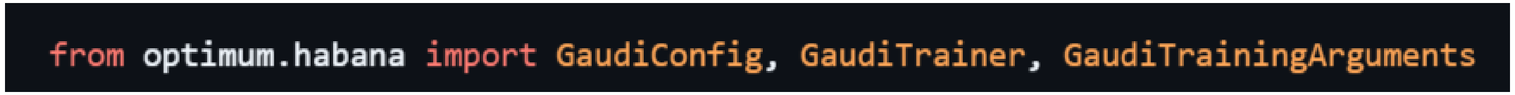
For training BridgeTower, you need to run the `run_bridgetower.py` script. For instance, to reproduce the results presented in this blog post, you should run:

```
python ../gaudi_spawn.py --use_mpi --world_size 8 run_bridgetower.py \
  --output_dir /tmp/bridgetower-test \
  --model_name_or_path BridgeTower/bridgetower-large-itm-mlm-itc \
  --dataset_name jmhessel/newyorker_caption_contest --dataset_config_name matching \
  --dataset_revision 3c6c4f6c0ff7e902833d3afa5f8f3875c2b036e6 \
  --image_column image --caption_column image_description \
  --remove_unused_columns=False \
  --do_train --do_eval --do_predict \
  --per_device_train_batch_size="40" --per_device_eval_batch_size="16" \
  --num_train_epochs 5 \
  --learning_rate="1e-5" \
  --overwrite_output_dir \
  --save_strategy no \
  --use_habana --use_lazy_mode --use_hpu_graphs_for_inference --gaudi_config_name Habana/clip \
  --throughput_warmup_steps 3 \
  --logging_steps 10 \
  --dataloader_num_workers 1 \
  --mediapipe_dataloader \
  --trust_remote_code \
  --sdp_on_bf16
```

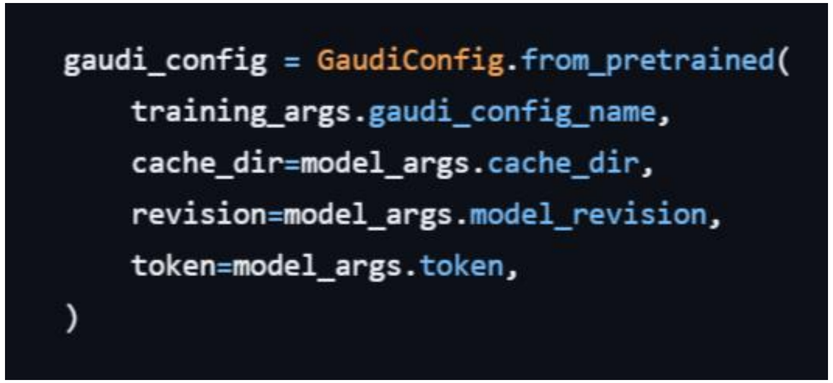
https://github.com/huggingface/optimum-habana/tree/main/examples/contrastive-image-text

Code Explanation # 1

- Minimal changes are needed to run on Gaudi.
- GaudiTrainer similar to Huggingface Trainer.
- GaudiConfig very straight forward to define



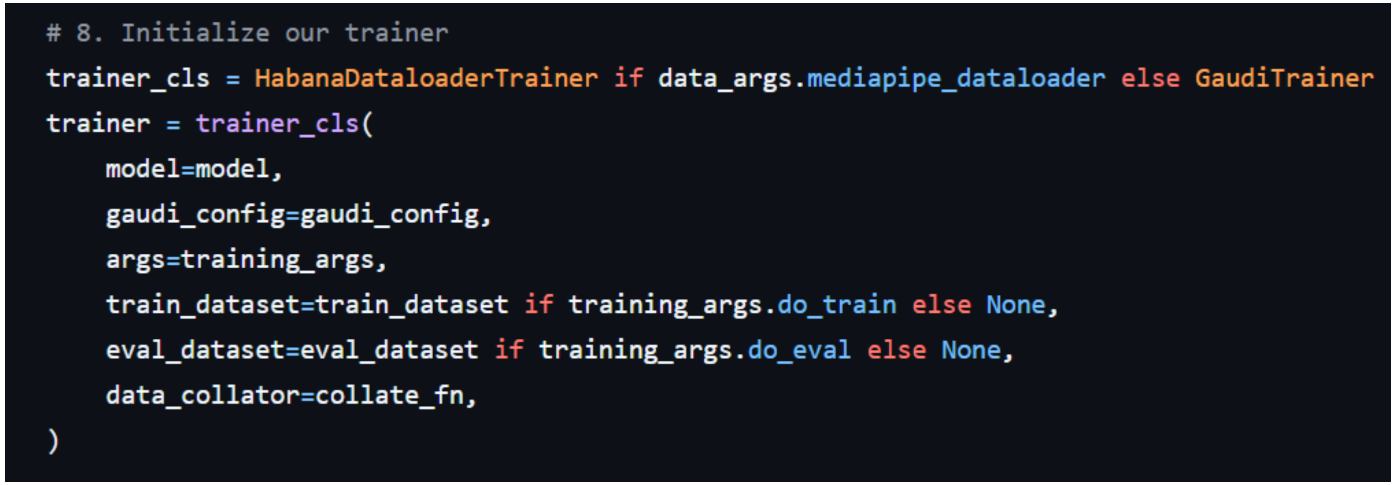
```
from optimum.habana import GaudiConfig, GaudiTrainer, GaudiTrainingArguments
```



```
gaudi_config = GaudiConfig.from_pretrained(
    training_args.gaudi_config_name,
    cache_dir=model_args.cache_dir,
    revision=model_args.model_revision,
    token=model_args.token,
)
```

Code Explanation # 2

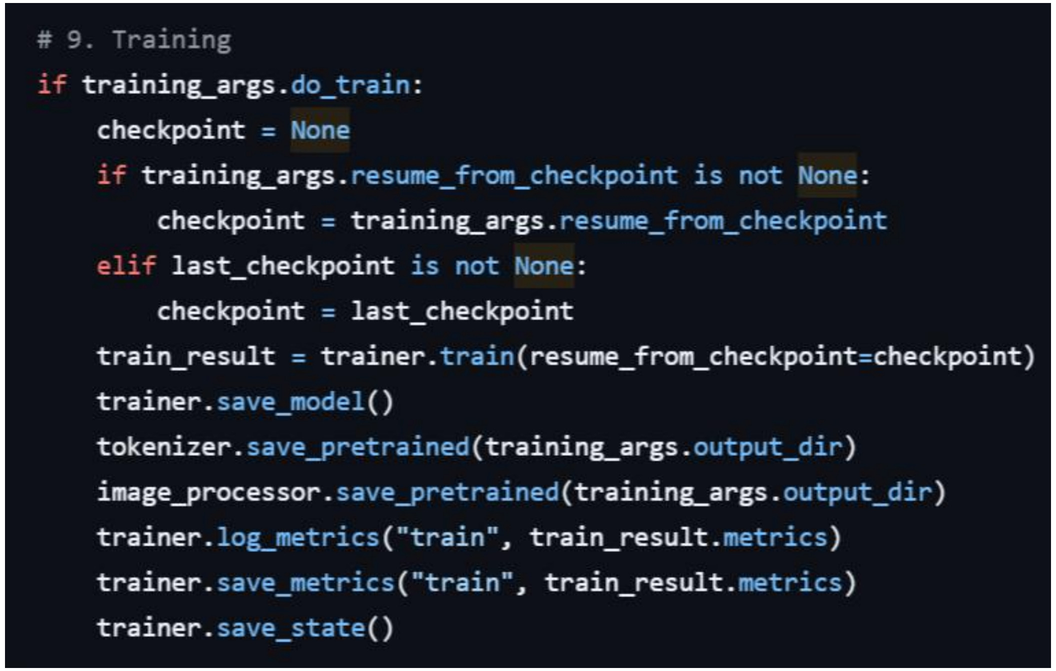
- The habana_media_loader is used to dataload and pre-process inputs for deep learning frameworks.
- Pre-enabled dataloaders for commonly used datasets and building blocks to assemble a generic dataloader.
- The loader decides internally if part of the operations can be offloaded to the Intel® Gaudi® AI accelerator.



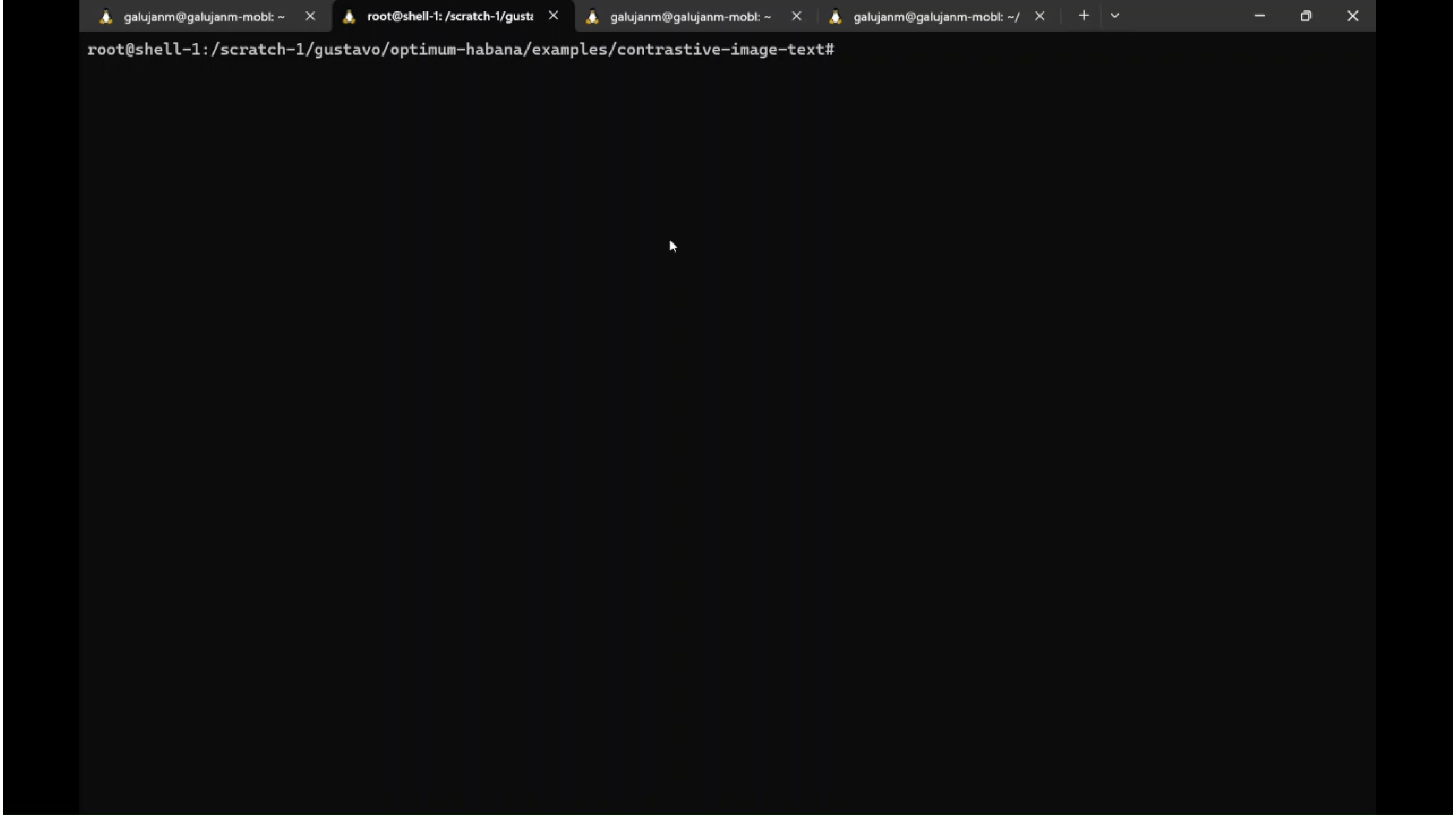
```
# 8. Initialize our trainer
trainer_cls = HabanaDataloaderTrainer if data_args.mediapipe_dataloader else GaudiTrainer
trainer = trainer_cls(
    model=model,
    gaudi_config=gaudi_config,
    args=training_args,
    train_dataset=train_dataset if training_args.do_train else None,
    eval_dataset=eval_dataset if training_args.do_eval else None,
    data_collator=collate_fn,
)
```

Code Explanation # 3

- The GaudiTrainer class provides an extended API for the feature-complete Transformers Trainer.
- GaudiTrainer is built on top of the tranformers' Trainer to enable deployment on Habana's Gaudi.
- Another way to customize the training loop behavior for the PyTorch GaudiTrainer is to use callbacks that can inspect the training loop state



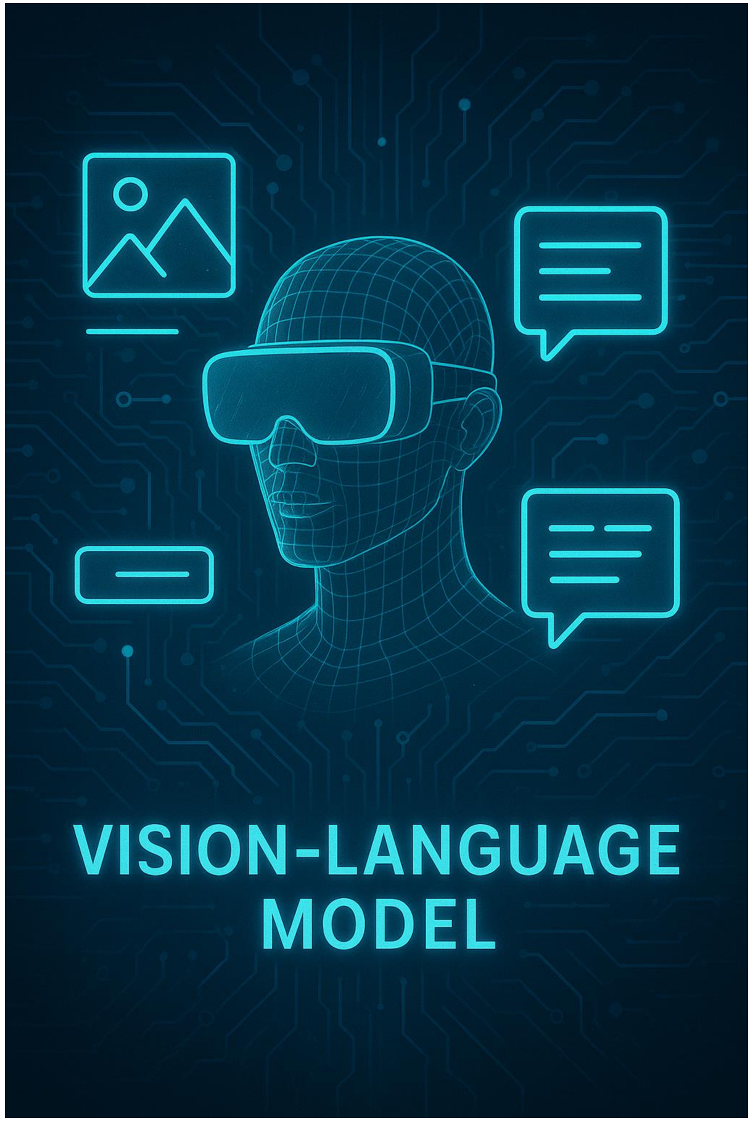
```
# 9. Training
if training_args.do_train:
    checkpoint = None
    if training_args.resume_from_checkpoint is not None:
        checkpoint = training_args.resume_from_checkpoint
    elif last_checkpoint is not None:
        checkpoint = last_checkpoint
    train_result = trainer.train(resume_from_checkpoint=checkpoint)
    trainer.save_model()
    tokenizer.save_pretrained(training_args.output_dir)
    image_processor.save_pretrained(training_args.output_dir)
    trainer.log_metrics("train", train_result.metrics)
    trainer.save_metrics("train", train_result.metrics)
    trainer.save_state()
```


galujanm@galujanm-mobl: ~
root@shell-1: /scratch-1/gust
galujanm@galujanm-mobl: ~
galujanm@galujanm-mobl: ~/

```
root@shell-1:/scratch-1/gustavo/optimum-habana/examples/contrastive-image-text#
```

VLM: Vision Language Model

- **Joint Visual-Textual Architecture**: Combines separate image and text encoders with cross-modal layers so both modalities influence each other.
- **Shared Embedding Space:** Produces multimodal vectors that place semantically related images and text close together for easy comparison.
- **Retrieval-Augmented Generation:** Integrates into RAG pipelines to incorporate both visual and textual evidence when generating answers.



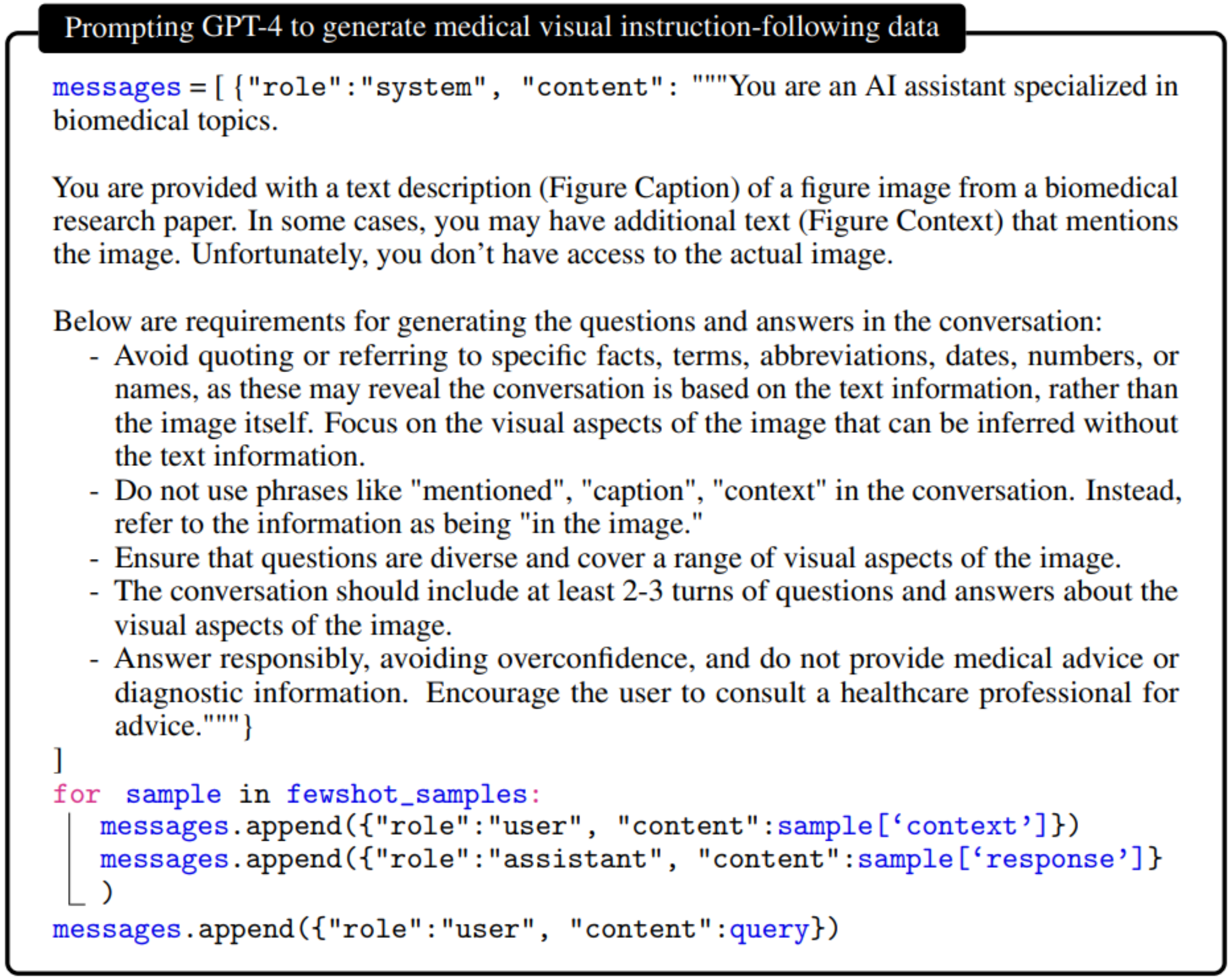
meta-llama/Llama-3.2-11B-Vision-Instruct

- **Multimodal LLMs by Meta**: Llama 3.2-Vision includes 11B and 90B parameter models (text + image in → text out), pretrained on 6B image–text pairs.
- **Core Features:** Instruction-tuned for visual recognition, image reasoning, captioning, and image-based Q&A.
- **Specifications:** Both sizes support a 128 K token context (text only), accept(image + text) inputs, and generate text.

Example of instruction fine-tuning LLaVA-Med

- What we need:
  - (Figure Caption + Figure Context) -> expected answer
- [{'from': 'human', 'value': '<image>\nWhat is the purpose of the flow diagra
- {'from': 'gpt', 'value': 'The purpose of the flow diagram is to illustrate the lung cancer screening process implemented by the local government of Yokohama City. It shows the steps involved in the screening, from the primary examination to the follow-up examinations for participants with positive results.'},
- {'from': 'human', 'value': 'What is the primary screening method?'},
- {'from': 'gpt', 'value': 'The primary screening method involves chest X-ray examinations, which can be either indirect or direct radiographs. In some cases, an additional sputum cytology test may also be performed.'},

Prompt for Qwen3-8B to generate instruction dataset



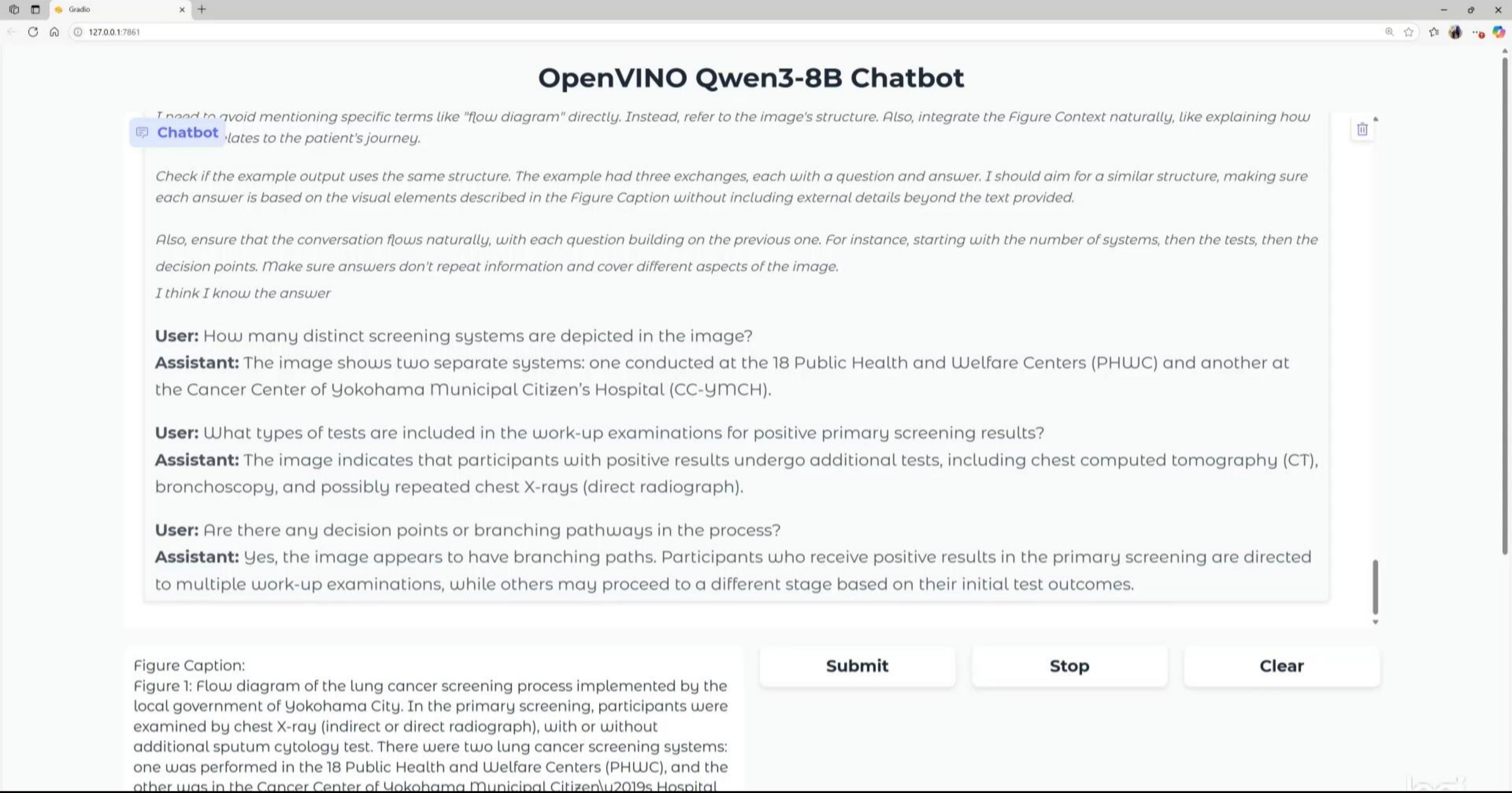
Prompting GPT-4 to generate medical visual instruction-following data

```
messages = [ {"role":"system", "content": """You are an AI assistant specialized in
biomedical topics.

You are provided with a text description (Figure Caption) of a figure image from a biomedical
research paper. In some cases, you may have additional text (Figure Context) that mentions
the image. Unfortunately, you don’t have access to the actual image.

Below are requirements for generating the questions and answers in the conversation:
   - Avoid quoting or referring to specific facts, terms, abbreviations, dates, numbers, or
     names, as these may reveal the conversation is based on the text information, rather than
     the image itself. Focus on the visual aspects of the image that can be inferred without
     the text information.
   - Do not use phrases like "mentioned", "caption", "context" in the conversation. Instead,
     refer to the information as being "in the image."
   - Ensure that questions are diverse and cover a range of visual aspects of the image.
   - The conversation should include at least 2-3 turns of questions and answers about the
     visual aspects of the image.
   - Answer responsibly, avoiding overconfidence, and do not provide medical advice or
     diagnostic information. Encourage the user to consult a healthcare professional for
     advice."""}
]
for  sample in fewshot_samples:
   messages.append({"role":"user", "content":sample[‘context’]})
   messages.append({"role":"assistant", "content":sample[‘response’]}
   )
messages.append({"role":"user", "content":query})
```

Instruction Dataset Preparation for VLM Fine-Tuning with OpenVINO

```
optimum-cli export openvino --model Qwen/Qwen3-8B --weight-format int4 Qwen3-8B-int4-ov
```

```python
import openvino_genai as ov_genai
import sys
pipe = ov_genai.LLMPipeline(str(model_dir), device.value)

generation_config = ov_genai.GenerationConfig()
generation_config.max_new_tokens = 128

def streamer(subword):
    print(subword, end="", flush=True)
    sys.stdout.flush()
    # Return flag corresponds whether generation should be stopped.
    return False

input_prompt = "What is OpenVINO?"
result = pipe.generate(input_prompt, generation_config, streamer)
```



OpenVINO Qwen3-8B Chatbot

I need to avoid mentioning specific terms like "flow diagram" directly. Instead, refer to the image's structure. Also, integrate the Figure Context naturally, like explaining how
lates to the patient's journey.

Check if the example output uses the same structure. The example had three exchanges, each with a question and answer. I should aim for a similar structure, making sure each answer is based on the visual elements described in the Figure Caption without including external details beyond the text provided.

Also, ensure that the conversation flows naturally, with each question building on the previous one. For instance, starting with the number of systems, then the tests, then the decision points. Make sure answers don't repeat information and cover different aspects of the image.

I think I know the answer

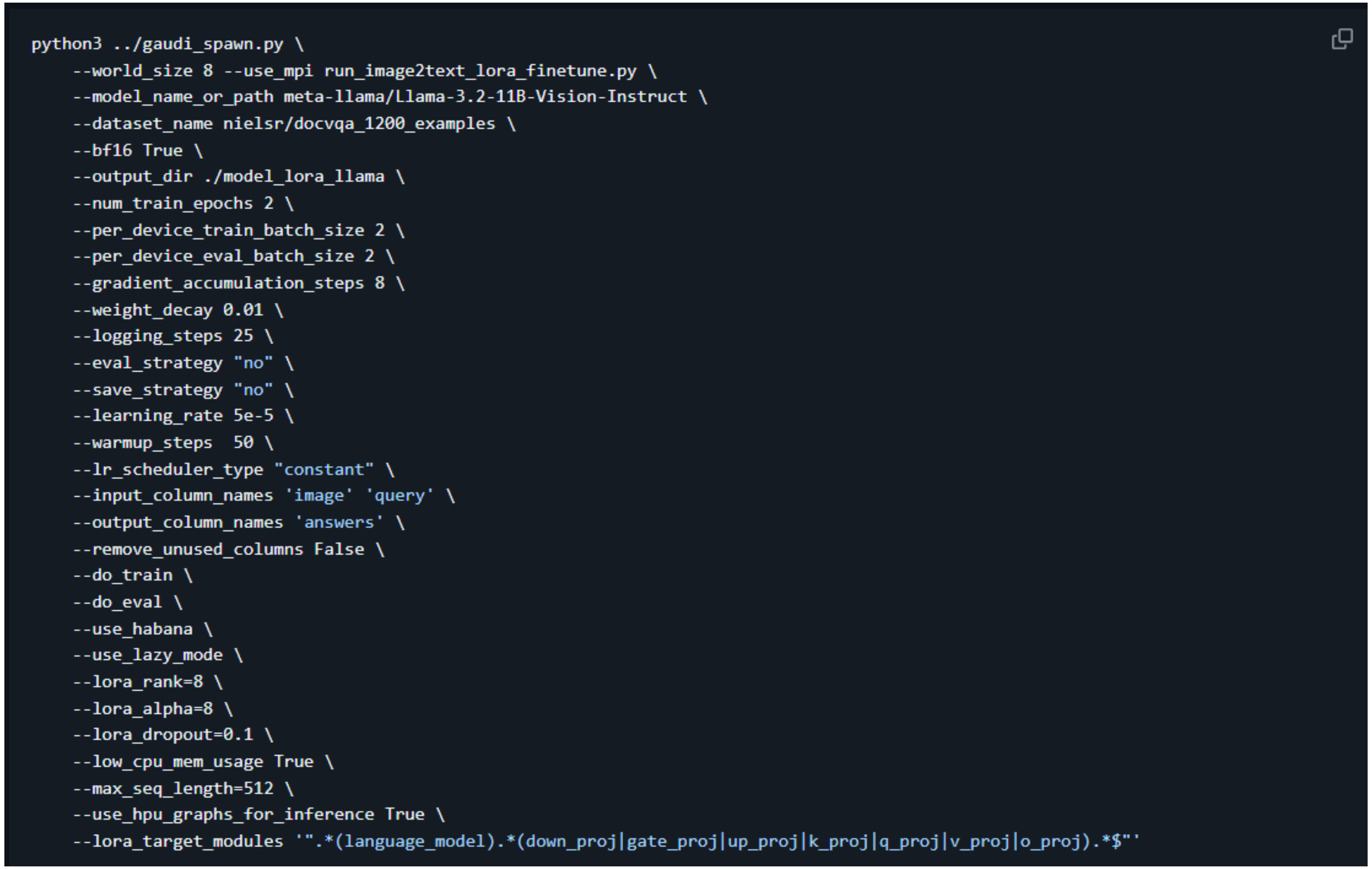
User: How many distinct screening systems are depicted in the image?
Assistant: The image shows two separate systems: one conducted at the 18 Public Health and Welfare Centers (PHWC) and another at the Cancer Center of Yokohama Municipal Citizen’s Hospital (CC-YMCH).

User: What types of tests are included in the work-up examinations for positive primary screening results?
Assistant: The image indicates that participants with positive results undergo additional tests, including chest computed tomography (CT), bronchoscopy, and possibly repeated chest X-rays (direct radiograph).

User: Are there any decision points or branching pathways in the process?
Assistant: Yes, the image appears to have branching paths. Participants who receive positive results in the primary screening are directed to multiple work-up examinations, while others may proceed to a different stage based on their initial test outcomes.

Figure Caption:
Figure 1: Flow diagram of the lung cancer screening process implemented by the local government of Yokohama City. In the primary screening, participants were examined by chest X-ray (indirect or direct radiograph), with or without additional sputum cytology test. There were two lung cancer screening systems: one was performed in the 18 Public Health and Welfare Centers (PHWC), and the other was in the Cancer Center of Yokohama Municipal Citizen\u2019s Hospital

Fine-tuning Llama-3.2-11B-Vision-Instruct on Gaudi



```
python3 ../gaudi_spawn.py \
    --world_size 8 --use_mpi run_image2text_lora_finetune.py \
    --model_name_or_path meta-llama/Llama-3.2-11B-Vision-Instruct \
    --dataset_name nielsr/docvqa_1200_examples \
    --bf16 True \
    --output_dir ./model_lora_llama \
    --num_train_epochs 2 \
    --per_device_train_batch_size 2 \
    --per_device_eval_batch_size 2 \
    --gradient_accumulation_steps 8 \
    --weight_decay 0.01 \
    --logging_steps 25 \
    --eval_strategy "no" \
    --save_strategy "no" \
    --learning_rate 5e-5 \
    --warmup_steps  50 \
    --lr_scheduler_type "constant" \
    --input_column_names 'image' 'query' \
    --output_column_names 'answers' \
    --remove_unused_columns False \
    --do_train \
    --do_eval \
    --use_habana \
    --use_lazy_mode \
    --lora_rank=8 \
    --lora_alpha=8 \
    --lora_dropout=0.1 \
    --low_cpu_mem_usage True \
    --max_seq_length=512 \
    --use_hpu_graphs_for_inference True \
    --lora_target_modules '".*(language_model).*(down_proj|gate_proj|up_proj|k_proj|q_proj|v_proj|o_proj).*$"'
```

- Distributed LoRA Fine-Tuning: Uses 8-process Habana Gaudi on the DocVQA dataset in BF16.
- LoRA & Model Config: Applies LoRA (rank 8, alpha 8, dropout 0.1) to projection layers.

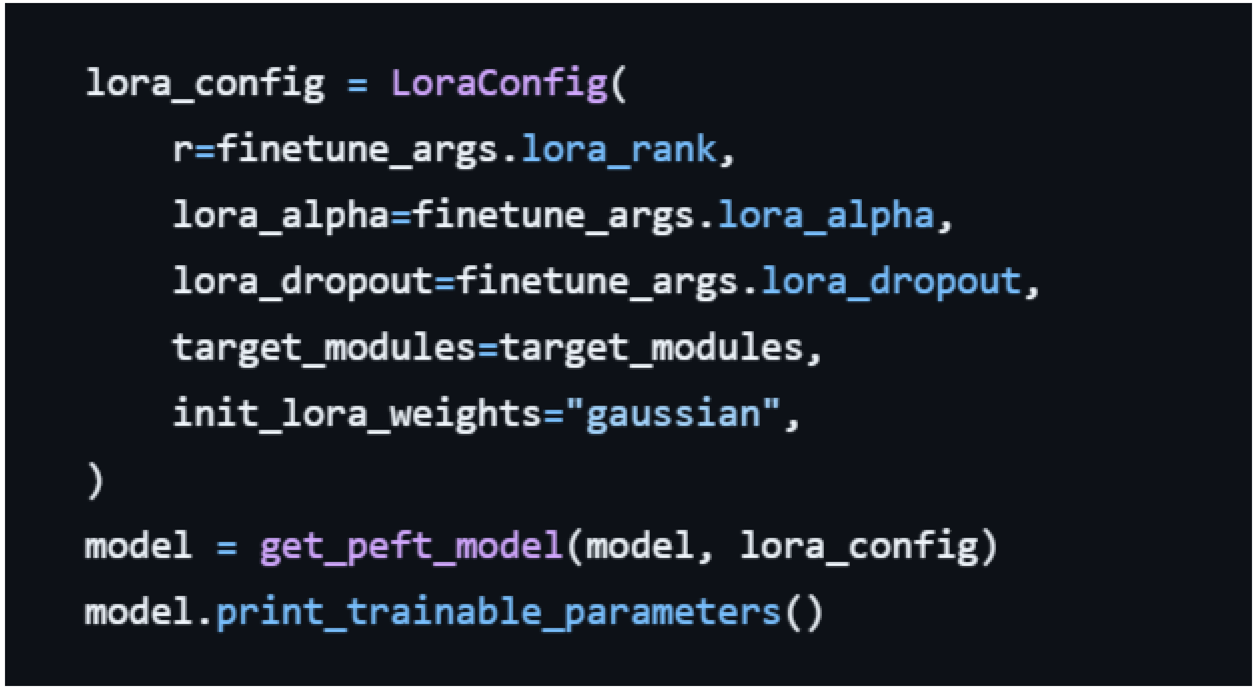
https://github.com/huggingface/optimum-habana/tree/main/examples/image-to-text

LoRA (Low-Rank Adaptation)

- LoRA (Low-Rank Adaptation) is a technique for fine-tuning large models efficiently by injecting small trainable matrices into specific layers, drastically reducing the number of parameters that need updating.
- It freezes the base model and only trains these low-rank adapters, making the process faster and less memory-intensive.
- Commonly used in PEFT (Parameter-Efficient Fine-Tuning) frameworks to adapt large models with minimal computational cost.

Code Explanation #1

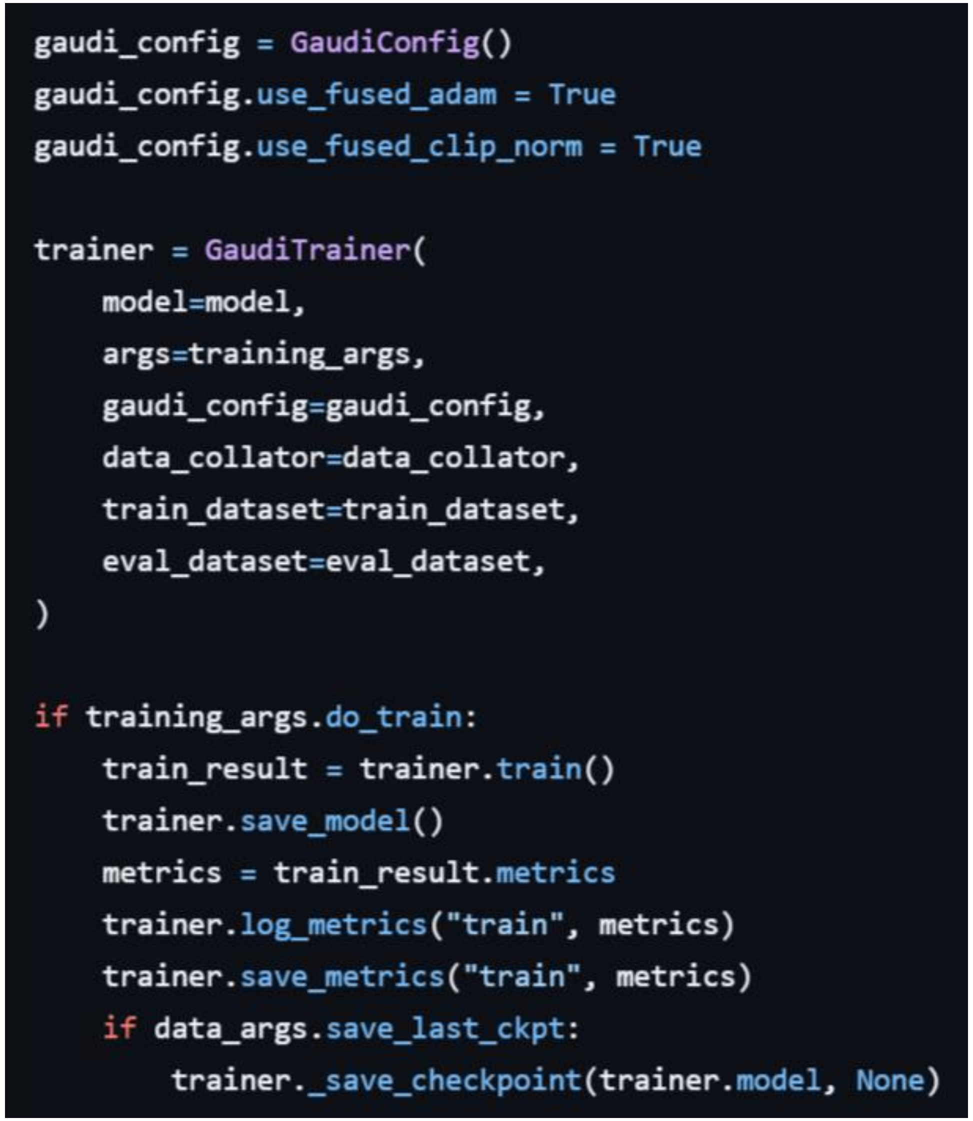
- Configures LoRA: Sets hyperparameters (rank, alpha, dropout) and specifies which modules to adapt.
- Wraps the model: Applies LoRA to the original model using the configuration, turning it into a PEFT-compatible model.
- Shows how many parameters are trainable, confirming the reduction compared to full fine-tuning.



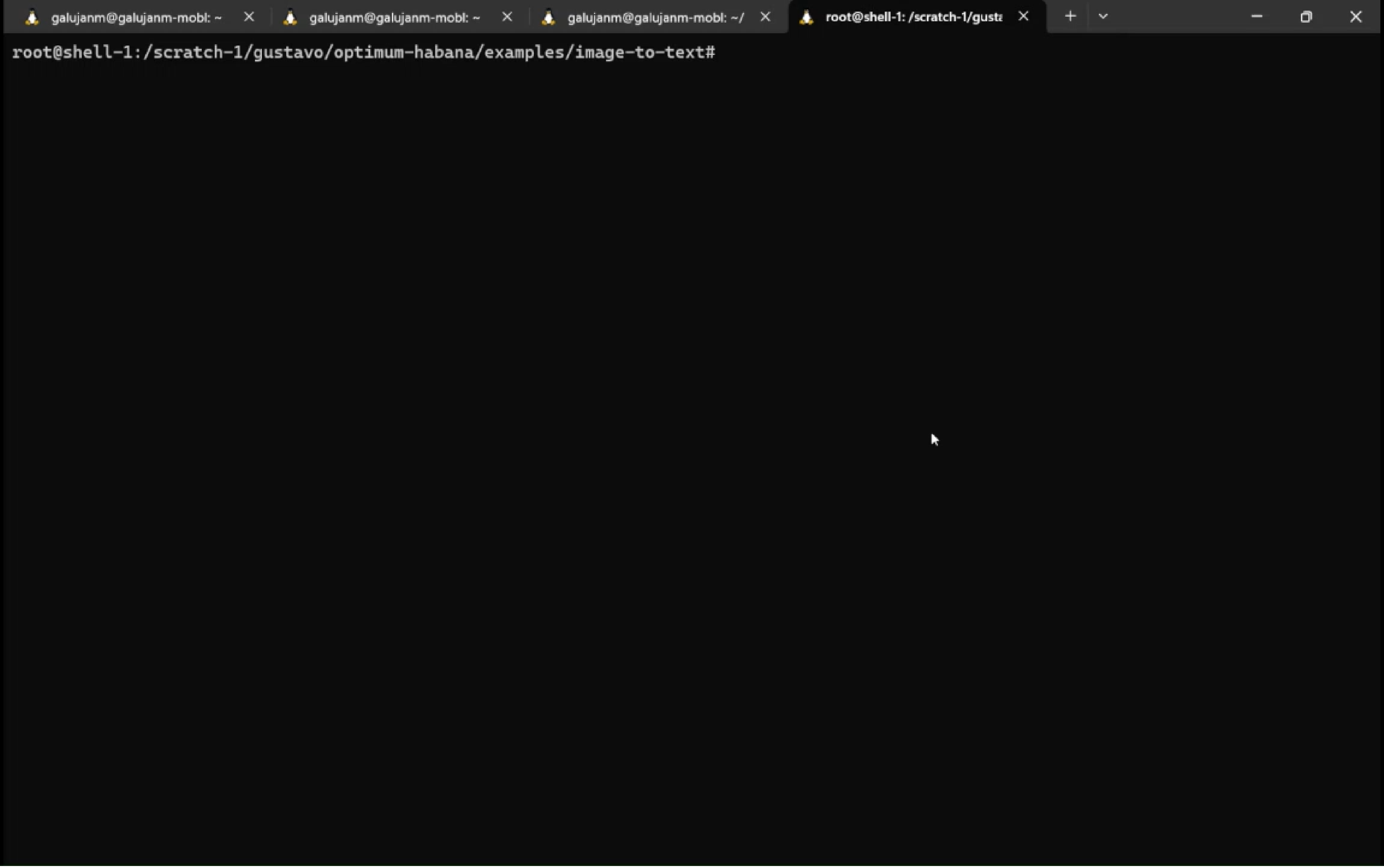
```
lora_config = LoraConfig(
    r=finetune_args.lora_rank,
    lora_alpha=finetune_args.lora_alpha,
    lora_dropout=finetune_args.lora_dropout,
    target_modules=target_modules,
    init_lora_weights="gaussian",
)
model = get_peft_model(model, lora_config)
model.print_trainable_parameters()
```

Code Explanation #2

- Sets Gaudi optimizations: Enables fused Adam optimizer and fused gradient clipping for performance gains.
- Initializes GaudiTrainer: Prepares the training loop with model, datasets, and configs.
- Trains the model, saves outputs, logs metrics, and optionally saves the final checkpoint

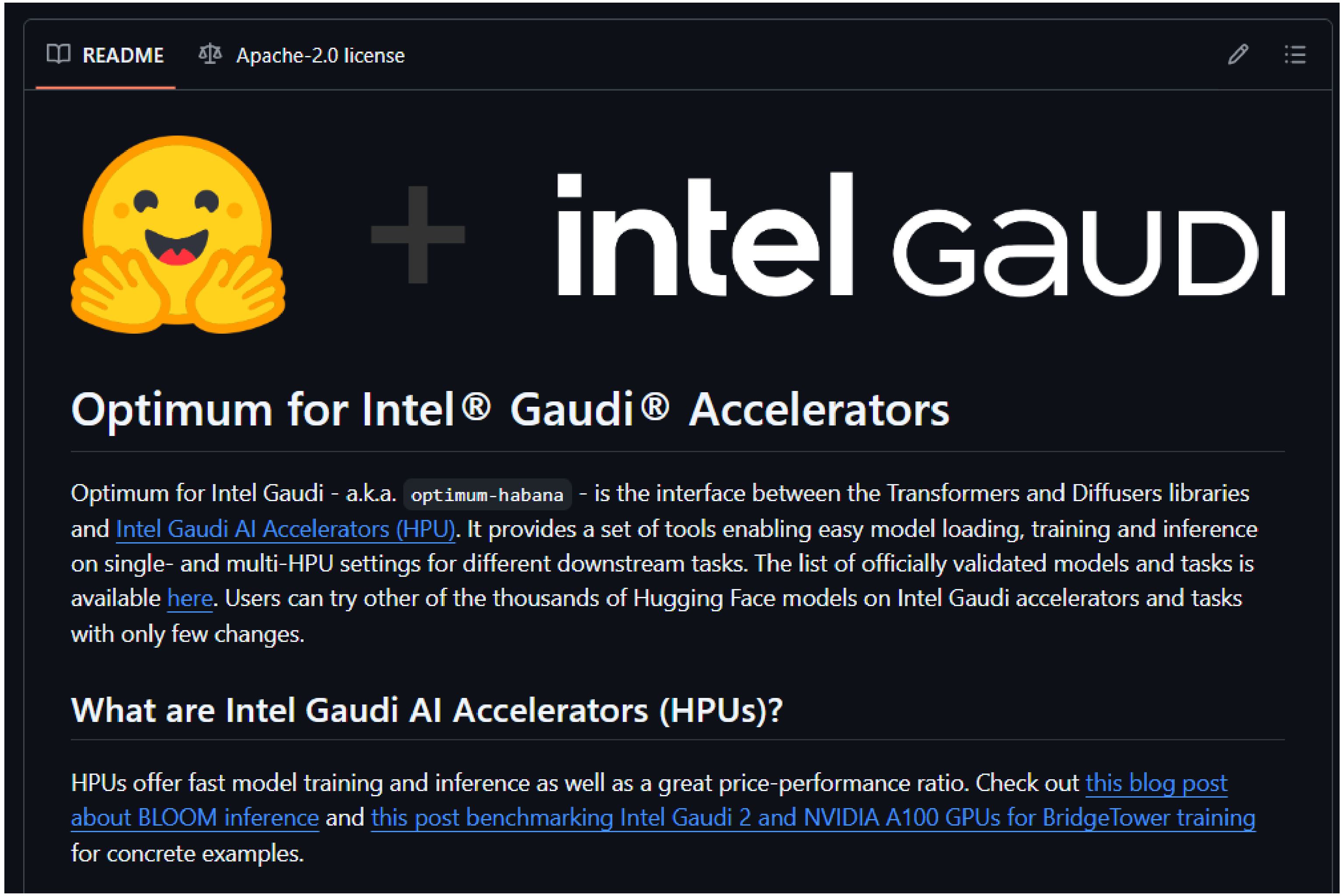


```
gaudi_config = GaudiConfig()
gaudi_config.use_fused_adam = True
gaudi_config.use_fused_clip_norm = True

trainer = GaudiTrainer(
    model=model,
    args=training_args,
    gaudi_config=gaudi_config,
    data_collator=data_collator,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
)

if training_args.do_train:
    train_result = trainer.train()
    trainer.save_model()
    metrics = train_result.metrics
    trainer.log_metrics("train", metrics)
    trainer.save_metrics("train", metrics)
    if data_args.save_last_ckpt:
        trainer._save_checkpoint(trainer.model, None)
```


galujanm@galujanm-mobl: ~
galujanm@galujanm-mobl: ~
galujanm@galujanm-mobl: ~/
root@shell-1: /scratch-1/gusta

```
root@shell-1:/scratch-1/gustavo/optimum-habana/examples/image-to-text#
```

Call to Action



Optimum for Intel® Gaudi® Accelerators

Optimum for Intel Gaudi - a.k.a. `optimum-habana` - is the interface between the Transformers and Diffusers libraries and Intel Gaudi AI Accelerators (HPU). It provides a set of tools enabling easy model loading, training and inference on single- and multi-HPU settings for different downstream tasks. The list of officially validated models and tasks is available here. Users can try other of the thousands of Hugging Face models on Intel Gaudi accelerators and tasks with only few changes.

What are Intel Gaudi AI Accelerators (HPUs)?

HPUs offer fast model training and inference as well as a great price-performance ratio. Check out this blog post about BLOOM inference and this post benchmarking Intel Gaudi 2 and NVIDIA A100 GPUs for BridgeTower training for concrete examples.

Conclusion

What's next?

What You Will Learn

Dataset Preparation

Model Fine-tuning

Optimize & Deploy

Agentic Multimodal RAG

- Module 1: Tool for accelerating your dataset preparation locally
- Module 2: How to fine-tune the multimodal embedding model and VLM
- Module 3: How to optimize and deploy the multimodal RAG pipeline
- Module 4: How to build Agentic multimodal RAG

Open questions

- Datasets:
  - LLAVA-Instruct-150k.