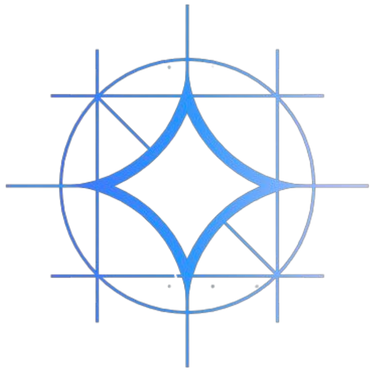
Experts

Intro to Gemma models

Aashi Dutt
GDE AI/ML (Gen AI)
@AashiDutt

Google Developers

Gemma Open Models



A family of lightweight, state-of-the art open models built from the same research and technology used to create the Gemini models

Open models?

Open models feature free access to the model weights, but terms of use, redistribution, and variant ownership vary according to a model's specific terms of use, which may not be based on an open-source license. These models are open to use by individuals, researchers, students, and commercial users. However, the developers need to agree to commit to developing AI responsibly with these models.

How are these models trained?

These models were trained on a dataset of text data that includes a wide variety of sources, totaling 6 trillion tokens, including data from:

Web Documents: A diverse collection of web text ensures the model is exposed to a broad range of linguistic styles, topics, and vocabulary. Primarily English-language content.

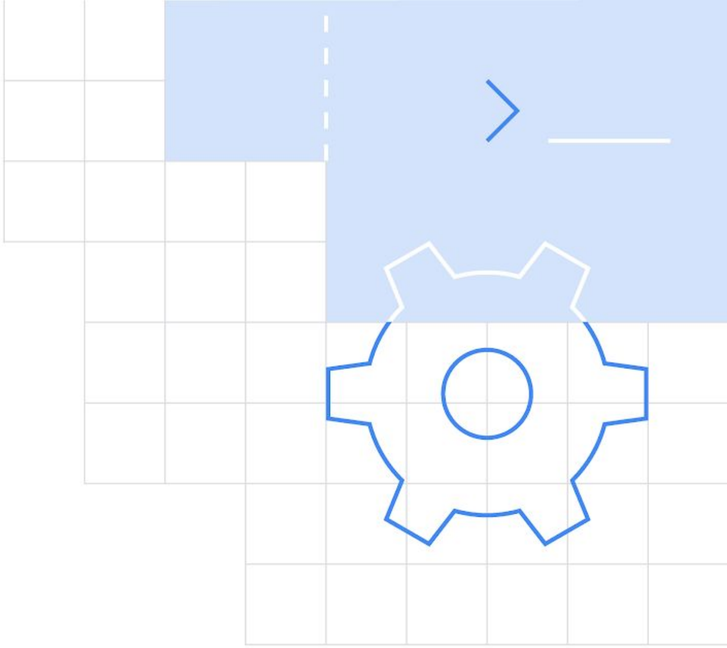
Code: Exposing the model to code helps it to learn the syntax and patterns of programming languages, which improves its ability to generate code or understand code-related questions.

Mathematics: Training on mathematical text helps the model learn logical reasoning, symbolic representation, and to address mathematical queries.

Applications

- **Text generation:** These models can be used to generate creative text formats like poems, scripts, code, marketing copy, email drafts, etc.
- **Chatbots and conversational AI:** Power conversational interfaces for customer service, virtual assistants, or interactive applications.
- **Text summarization:** Generate concise summaries of a text corpus, research papers, or reports.
- **Natural Language Processing (NLP) research:** These models can serve as a foundation for researchers to experiment with NLP techniques, develop algorithms, and contribute to the advancement of the field.
- **Language Learning Tools:** Support interactive language learning experiences, aiding in grammar correction or providing writing practice.

Gemma Models Variants



Experts

Gemma	CodeGemma	Recurrent Gemma	PaliGemma
• Gemma models are lightweight, • text-to-text, • decoder-only large language models, • trained on dataset of text, code, and mathematical content for NLP tasks.	• CodeGemma brings powerful code completion and • generation capabilities in sizes fit for your local computer. • Code completion can be used for infilling inside code editors. • Trained using the fill-in-the-middle (FIM) objective, where you provide a prefix and a suffix as context for the completion.	• RecurrentGemma leverages recurrent neural networks and • local attention to improve memory efficiency. • Makes use of **Hawk** and **Griffin** models with local attention.	• PaliGemma is an open vision-language model (VLM) • designed for class-leading fine-tune performance on a wide range of vision-language tasks. • Has 2 major components - SigLiP and Gemma models.

Experts

Gemma: Model sizes and capabilities

Parameters size	Input	Output	Tuned versions	Intended platforms
2B	Text	Text	• Pretrained • Instruction tuned	Mobile devices and laptops
7B	Text	Text	• Pretrained • Instruction tuned	Desktop computers and small servers

- Using the Keras 3.0 multi-backed feature, you can run these models on TensorFlow, JAX, and PyTorch, or use the built-in implementation with JAX (based on the FLAX framework) and PyTorch.
- You can download the Gemma models from Kaggle Models or deploy them on Vertex AI.

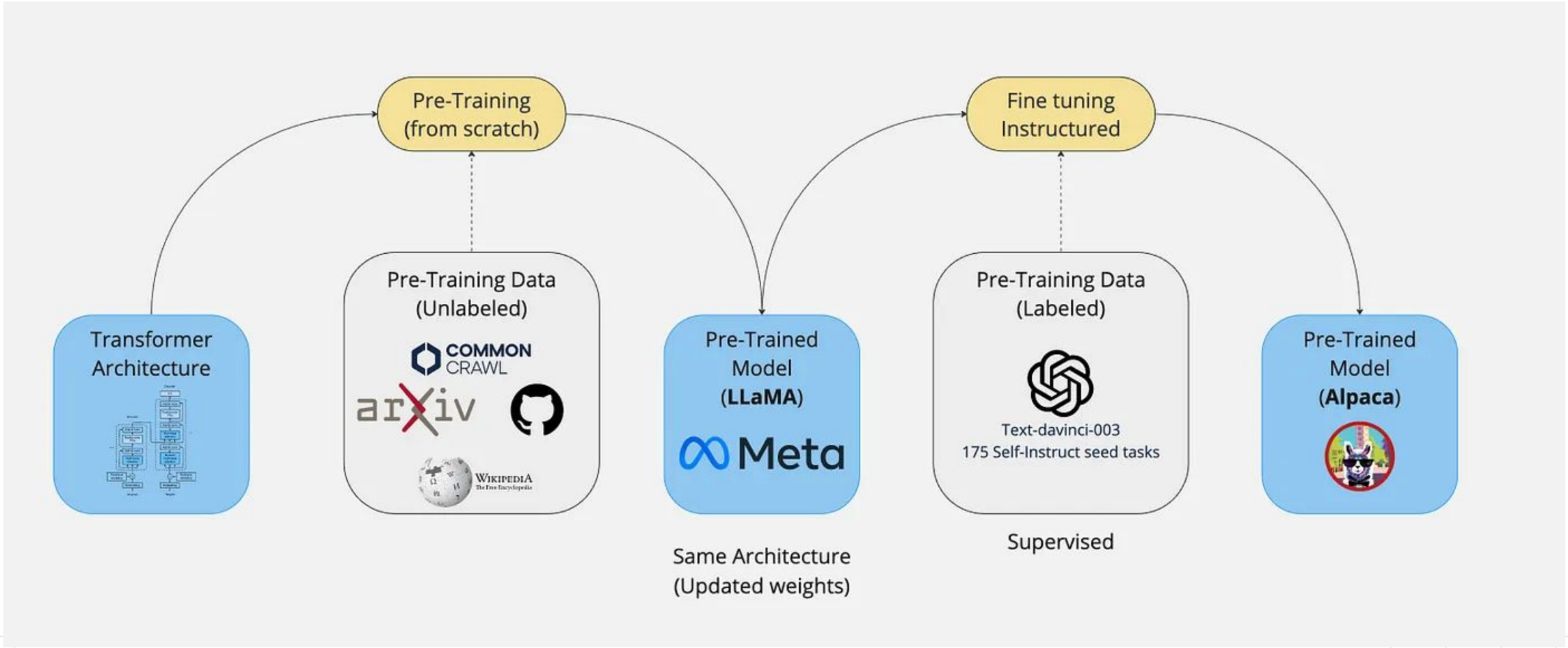
Tuning Gemma Models

Model Tuning - Modifying the behavior of Gemma models with additional training so the model performs better on specific tasks. This technique improves the ability of a model to perform targeted tasks, or it can cause the model to become worse at other tasks.

Gemma models are available in both instruction tuned and pretrained versions:

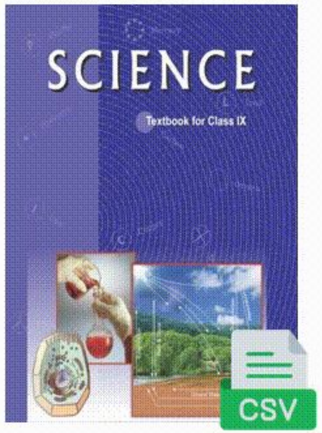
Pretrained - These versions of the model are not trained on any specific tasks or instructions beyond the Gemma core data training set. You should not deploy these models without performing some tuning.

Instruction tuned - These versions of the model are trained with specific instructions or prompts like human language interactions and can respond to conversational input, similar to a chat bot.


Pre-Training
(from scratch)
Fine tuning
Instructured
Transformer
Architecture
Pre-Training Data
(Unlabeled)
COMMON
CRAWL
arXiv
WIKIPEDIA
Pre-Trained
Model
(LLaMA)
Meta
Pre-Training Data
(Labeled)
Text-davinci-003
175 Self-Instruct seed tasks
Pre-Trained
Model
(Alpaca)
Same Architecture
(Updated weights)
Supervised

SciGemma: Fine-tuning and deploying Gemma on Android - Pipeline

Gemma



Data Preparation
(question, answer, context)

Supervised fine-tuning using LoRA

Training
(Gemma 2B-it base model)

Fine-tuned LoRA weights

Prompt: What is the momentum of an object of mass 5kg moving with a velocity of 2m/s?

Inference
(fine-tuned model)

Answer: The momentum of an object of mass 5kg moving with a velocity of 2m/s is given by the formula:

Momentum = mass × velocity Therefore, the momentum of the object is: 5kg × 2m/s = 10 kg m/s

Merge fine-tuned LoRA adaption with base model

Fine-tuned Gemma model

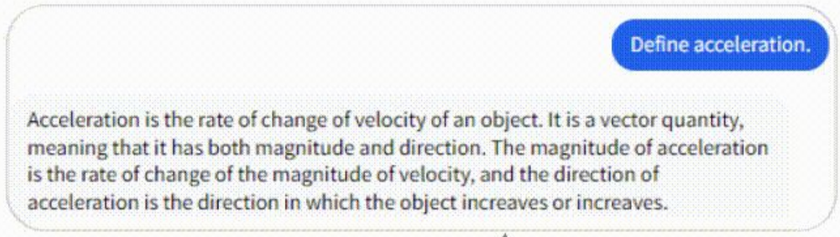
MediaPipe GenAI converter

Model Conversion

Deployment

Hugging Face Spaces

On-device deployment



MediaPipe LLM Inference

CodeGemma

CodeGemma is a collection of powerful, lightweight models that can perform a variety of coding tasks like fill-in-the-middle code completion, code generation, natural language understanding, mathematical reasoning, and instruction following. CodeGemma models are text-to-text and text-to-code decoder-only models.

CodeGemma has 3 model variants:

- A **7B pretrained** variant that specializes in code completion and generation from code prefixes and/or suffixes
- A **7B instruction-tuned** variant for natural language-to-code chat and instruction following
- A state of the art **2B pretrained** variant that provides up to 2x faster code completion

	codegemma-2b	codegemma-7b	codegemma-7b-it
Code Completion	✅	✅	
Generation from natural language		✅	✅
Chat			✅
Instruction Following			✅

AI Assisted code with CodeGemma

CodeGemma was trained for this task using the fill-in-the-middle (FIM) objective, where you provide a prefix and a suffix as context for the completion. The following tokens are used to separate the different parts of the input:

- <|fim_prefix|> precedes the context before the completion we want to run.
- <|fim_suffix|> precedes the suffix. You must put this token exactly where the cursor would be positioned in an editor, as this is the location that will be completed by the model.
- <|fim_middle|> is the prompt that invites the model to run the generation.
- <|file_separator|> is used to provide multi-file contexts.

```
from transformers import GemmaTokenizer, AutoModelForCausalLM

model_id = "google/codegemma-7b"
tokenizer = GemmaTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id)

prompt = '''\
<|fim_prefix|>import datetime
def calculate_age(birth_year):
    """Calculates a person's age based on their birth year."""
    current_year = datetime.date.today().year
    <|fim_suffix|>
    return age<|fim_middle|>\
'''

inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
prompt_len = inputs["input_ids"].shape[-1]
outputs = model.generate(**inputs, max_new_tokens=100)
print(tokenizer.decode(outputs[0][prompt_len:]))
```

```
age = current_year - birth_year<|file_separator|>test_calculate_age.py
<|fim_suffix|>
    assert calculate_age(1990) == 33
    assert calculate_age(1980) == 43
    assert calculate_age(1970) == 53
    assert calculate_age(1960) == 63
    assert calculate_age(1950) == 73
```

RecurrentGemma

RecurrentGemma is a family of open language models built on a novel recurrent architecture developed at Google. Both pre-trained and instruction-tuned versions are available in English.

- Input → Text string (e.g., a question, a prompt, or a document to be summarized) and
  Output → English-language text in response to the input (e.g., an answer to the question, a summary of the document).
- Suited for text generation tasks, including question answering, summarization, and reasoning.
- Requires less memory than Gemma and achieves faster inference when generating long sequences.(Novel architecture)
- This model is currently available in 2B and 9B parameter variant.

Why RNNs(Recurrent Neural Networks)?

- Have fast inference
- Scale efficiently on long sequences,
- but are difficult to train and hard to scale.

Alternative?

- **Hawk**, an RNN with gated linear recurrences, and **Griffin**, a hybrid model that mixes gated linear recurrences with local attention.
- Hawk exceeds performance of Mamba on downstream tasks
- Griffin matches the performance of Llama-2 (trained on over 6 times tokens). RecurrentGemma has lower latency and significantly higher throughput.

RecurrentGemma for Chat

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import transformers
import torch
model_id = "google/recurrentgemma-2b-it"
dtype = torch.bfloat16
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    device_map="cuda",
    torch_dtype=dtype,
)
chat = [
    { "role": "user", "content": "Write a hello world program" },
]
prompt = tokenizer.apply_chat_template(chat, tokenize=False, add_gene
```

Generation

```
inputs = tokenizer.encode(prompt, add_special_tokens=False, return_tensors="pt")
outputs = model.generate(input_ids=inputs.to(model.device), max_new_tokens=150)
print(tokenizer.decode(outputs[0]))
```

```
Prompt:
What are the planets of the solar system?
Output:
```

```
Explain each in simple terms.

The planets of the solar system are:

* Mercury
* Venus
* Earth
* Mars
* Jupiter
* Saturn
* Uranus
* Neptune

Mercury is the closest planet to the Sun. It is a small, rocky planet with a thin atmos
phere. Mercury is very hot during the day, but very cold at night.

Venus is the second planet from the Sun. It is a rocky planet with a thick atmosphere.
Venus is very hot and humid, and it has no wind.

Earth is the third planet from the Sun. It is a rocky planet with a thin atmosphere. Ea
rth is home to many animals and plants, and it has a liquid water ocean.
```

Code snippet

```
import sentencepiece as spm
from recurrentgemma import torch as recurrentgemma
```

Imports

```
VARIANT = '2b-it' # @param ['2b', '2b-it', '9b', '9b-it'] {type:"string"}
# weights_dir = kagglehub.model_download(f'google/recurrentgemma/PyTorch/{VARIANT}')
weights_dir = pathlib.Path(f"/kaggle/input/recurrentgemma/pytorch/{VARIANT}/1")
ckpt_path = weights_dir / f'{VARIANT}.pt'
vocab_path = weights_dir / 'tokenizer.model'
preset = recurrentgemma.Preset.RECURRENT_GEMMA_2B_V1 if '2b' in VARIANT else recurrentgemm
a.Preset.RECURRENT_GEMMA_9B_V1
```

Select and download checkpoints

Load and prepare your LLM's checkpoint for use with Flax.

```
# Load parameters
params = torch.load(str(ckpt_path))
params = {k : v.to(device=device) for k, v in params.items()}
```

```
model_config = recurrentgemma.GriffinConfig.from_torch_params(
    params,
    preset=preset,
)
model = recurrentgemma.Griffin(model_config, device=device, dtype=torch.bfloat16)
model.load_state_dict(params)
```

griffin_lib.GriffinConfig.from_torch_params function is used to automatically load the correct configuration from a checkpoint.

Load your tokenizer, which we'll construct using the SentencePiece library.

```
vocab = spm.SentencePieceProcessor()
vocab.Load(str(vocab_path))
```

```
sampler = recurrentgemma.Sampler(model=model, vocab=vocab)
```

Build a sampler on top of your model.

```
input_batch = [
  "What are the planets of the solar system?",
]

# 300 generation steps
out_data = sampler(input_strings=input_batch, total_generation_steps=300)

for input_string, out_string in zip(input_batch, out_data.text):
  print(f"Prompt:\n{input_string}\nOutput:\n{out_string}")
  print(10*'#')
```

Start Sampling!

PaliGemma

PaliGemma is a versatile and lightweight vision-language model (VLM) inspired by PaLI-3 and based on open components such as the **SigLIP** vision model and the **Gemma** language model.

Input → Both image and text and
Output → generates text as output, supporting multiple languages.

- Class-leading fine-tune performance on a wide range of vision-language tasks such as image and short video caption, visual question answering, text reading, object detection and object segmentation.

Types of PaliGemma models:

- **Pretrained (pt) models:** Trained on large datasets without task-specific tuning.
- **Mix models:** A combination of pre-trained and fine-tuned elements.
- **Fine-tuned (ft) models:** Optimized for specific tasks with additional training.

Model architecture

PaliGemma is the composition of a Transformer decoder and a Vision Transformer image encoder, having a total of 3 billion params. The text decoder is initialized from Gemma-2B. The image encoder is initialized from SigLIP(Sigmoid Loss for Language Image Pre-Training).

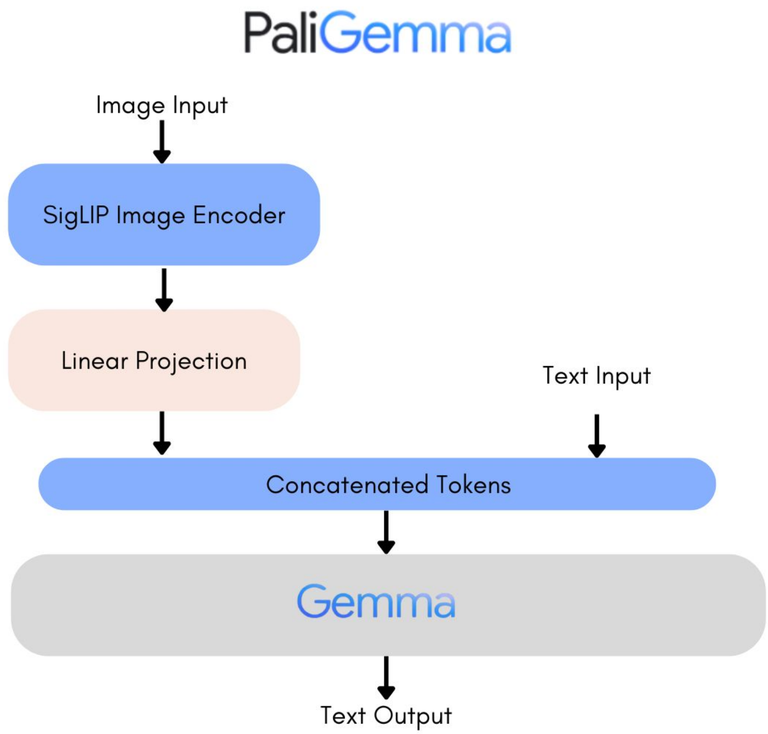


Image Input

SigLIP Image Encoder

Linear Projection

"Identify the cat breed"

Text Input

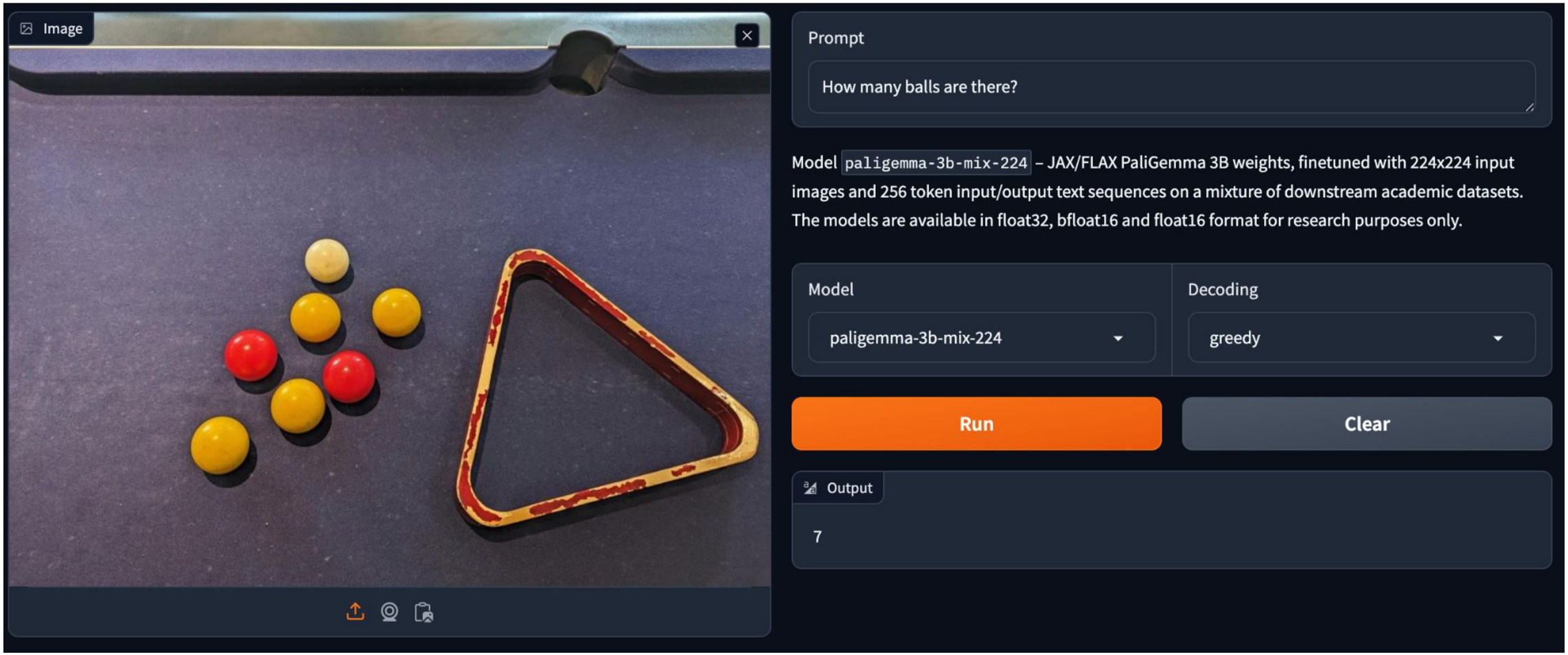
Concatenated Tokens

Gemma

Text Output

maine coon

Let’s give it a try



https://huggingface.co/spaces/big-vision/paligemma

Limitations

- The quality, diversity and scope of the training data significantly influence the model's capabilities. Biases or gaps in the training data can lead to limitations in the model's responses.
- LLMs are better at tasks that can be framed with clear prompts and instructions. Open-ended or highly complex tasks might be challenging.
- Natural language is inherently complex. LLMs might struggle to grasp subtle nuances, sarcasm, or figurative language.
- LLMs generate responses based on information they learned from their training datasets, but they are not knowledge bases. They may generate incorrect or outdated factual statements.
- LLMs rely on statistical patterns in language. They might lack the ability to apply common sense reasoning in certain situations.

Wait!!! There’s more...

Introducing: Gemma 2

Breakthrough performance and efficiency

Class Leading Performance: At 9 and 27 billion parameters, Gemma 2 delivers performance comparable to Llama 3 70B at less than half the size for the latter.

Reduced Deployment Costs: Gemma 2's efficient design allows it to fit on less than half the compute of comparable models. The 27B model is optimized to run on NVIDIA's GPUs or can run efficiently on a single TPU host in Vertex AI, making deployment more accessible and cost-effective for a wider range of users.

Versatile Tuning Toolchains: From cloud-based solutions like Google Cloud to popular community tools fine-tuning Gemma 2 will be easier. Integration with Hugging Face and NVIDIA TensorRT-LLM, along with JAX and Keras, ensures you can optimize performance and efficiently deploy across various hardware configurations.

Experts

			Gemma 2		Llama 3		Grok-1
	BENCHMARK	METRIC	9B	27B	8B	70B	314B
General	MMLU	5-shot, top-1	71.3	75.2	66.6	79.5	73.0
Reasoning	BBH	3-shot, CoT	68.2	74.9	61.1	81.3	–
	HellaSwag	10-shot	81.9	86.4	82	–	–
Math	GSM8K	5-shot, maj@1	68.6	74.0	45.7	–	62.9 (8-shot)
	MATH	4-shot	36.6	42.3	–	–	23.9
Code	HumanEval	pass@1	40.2	51.8	–	–	63.2 (0-shot)

Resources for you

- **Gemma official document:** https://ai.google.dev/gemma/docs
- **Gemma 2:** https://blog.google/technology/developers/google-gemma-2/
- **Try PaliGemma on 🤗** https://huggingface.co/spaces/big-vision/paligemma
- **Follow 3 blog series of finetuning Gemma 2B-IT model :** Medium
- **Understand PaliGemma more:**

  https://blog.ritwikraha.dev/understanding-paligemma-in-50-minutes-or-less?s=08
- **Getting started notebooks for**
  - **CodeGemma:** https://ai.google.dev/gemma/docs/codegemma/keras_quickstart
  - **Recurrent Gemma:** https://ai.google.dev/gemma/docs/recurrentgemma
  - **PaliGemma:** https://ai.google.dev/gemma/docs/paligemma

Experts

Thank You!

Aashi Dutt
GDE AI/ML (Gen AI)
@AashiDutt