TxGemma model card

Model documentation: TxGemma (https://developers.google.com/health-ai-developer-foundations/txgemma)

Resources:

- Model on Google Cloud Model Garden: TxGemma (https://console.cloud.google.com/vertex-ai/publishers/google/model-garden/txgemma)
- Model on Hugging Face: TxGemma (https://huggingface.co/collections/google/txgemma-release-67dd92e931c857d15e4d1e87)
- GitHub repository (supporting code, Colab notebooks, discussions, and issues): TxGemma (https://github.com/google-gemini/gemma-cookbook/tree/main/TxGemma)
- Quick start notebook: notebooks/quick_start (https://github.com/google-gemini/gemma-cookbook/blob/main/TxGemma/%5BTxGemma%5DQuickstart_with_Hugging_Face.ipynb)
- Support: See Contact (https://developers.google.com/health-ai-developer-foundations/txgemma/get-started.md#contact).

Terms of use: Health AI Developer Foundations terms of use (https://developers.google.com/health-ai-developer-foundations/terms)

Author: Google

Model information

This section describes the TxGemma model and how to use it.

Description

TxGemma is a collection of lightweight, state-of-the-art, open language models built upon Gemma 2, fine-tuned for therapeutic development. It comes in 3 sizes, 2B, 9B, and 27B.

TxGemma models are designed to process and understand information related to various therapeutic modalities and targets, including small molecules, proteins, nucleic acids, diseases, and cell lines. TxGemma excels at tasks such as property prediction, and can serve as a foundation for further fine-tuning or as an interactive, conversational agent for drug discovery. The model is fine-

tuned from Gemma 2 using a diverse set of instruction-tuning datasets, curated from the Therapeutics Data Commons (TDC) (https://tdcommons.ai/).

TxGemma is offered as both a prediction model that expects a narrow form of prompting and for the 9B and 27B version, conversational models that are more flexible and can be used in multi-turn interactions, including to explain its rationale behind a prediction. This conversational model comes at the expense of some raw prediction performance. See our manuscript (https://arxiv.org/abs/2504.06196) for more information.

Key Features

- Versatility: Exhibits strong performance across a wide range of therapeutic tasks, outperforming or matching best-in-class performance on a significant number of benchmarks.
- Data Efficiency: Shows competitive performance even with limited data compared to larger models, offering improvements over its predecessors.
- Conversational Capability (TxGemma-Chat): Includes conversational variants that can engage in natural language dialogue and explain the reasoning behind their predictions.
- Foundation for Fine-tuning: Can be used as a pre-trained foundation for specialized use cases.

Potential Applications

TxGemma can be a valuable tool for researchers in the following areas:

- Accelerated Drug Discovery: Streamline the therapeutic development process by predictin properties of therapeutics and targets for a wide variety of tasks including target identifica drug-target interaction prediction, and clinical trial approval prediction.

How to use

Below are some example code snippets to help you quickly get started running the model locally on GPU. If you want to use the model to run inference on a large number of inputs, we recommend that you create a production version using Model Garden (https://console.cloud.google.com/vertex-ai/publishers/google/model-garden/txgemma).

Formatting prompts for therapeutic tasks

```
import json
from huggingface_hub import hf_hub_download
```

```
# Load prompt template for tasks from TDC
tdc_prompts_filepath = hf_hub_download(
    repo_id="google/txgemma-27b-predict",
    filename="tdc_prompts.json",
)
with open(tdc_prompts_filepath, "r") as f:
    tdc_prompts_json = json.load(f)

# Set example TDC task and input
task_name = "BBB_Martins"
input_type = "{Drug SMILES}"
drug_smiles = "CN1C(=O)CN=C(C2=CCCCC2)c2cc(Cl)ccc21"

# Construct prompt using template and input drug SMILES string
TDC_PROMPT = tdc_prompts_json[task_name].replace(input_type, drug_smiles)
print(TDC_PROMPT)
```

✦Code Tutor

The resulting prompt is in the format expected by the model:

```
Instructions: Answer the following question about drug properties.
Context: As a membrane separating circulating blood and brain extracellular fluid, th
Question: Given a drug SMILES string, predict whether it
(A) does not cross the BBB (B) crosses the BBB
Drug SMILES: CN1C(=O)CN=C(C2=CCCCC2)c2cc(Cl)ccc21
Answer:
```

Can I help?

Running the model on predictive tasks

```
# pip install accelerate transformers
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load model directly from Hugging Face Hub
tokenizer = AutoTokenizer.from_pretrained("google/txgemma-27b-predict")
model = AutoModelForCausalLM.from_pretrained(
    "google/txgemma-27b-predict",
    device_map="auto",
)

# Formatted TDC prompt (see "Formatting prompts for therapeutic tasks" section above)
```

```
prompt = TDC_PROMPT

# Prepare tokenized inputs
input_ids = tokenizer(prompt, return_tensors="pt").to("cuda")

# Generate response
outputs = model.generate(**input_ids, max_new_tokens=8)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

✦Code Tutor

Alternatively, you can use the **`pipeline`** API, which provides a simple way to run inference while abstracting away complex details of loading and using the model and tokenizer:

```
# pip install transformers
from transformers import pipeline

# Instantiate a text generation pipeline using the model
pipe = pipeline(
    "text-generation",
    model="google/txgemma-27b-predict",
    device="cuda",
)

# Formatted TDC prompt (see "Formatting prompts for therapeutic tasks" section a
prompt = TDC_PROMPT

# Generate response
outputs = pipe(prompt, max_new_tokens=8)
response = outputs[0]["generated_text"]
print(response)
```

✦Code Tutor

Can I help?

Applying the chat template for conversational use

TxGemma-Chat models use a chat template that must be adhered to for conversational use. The easiest way to apply it is using the tokenizer`s built-in chat template, as shown in the following snippet.

Let's load the model and apply the chat template to a conversation. In this example, we'll start with a single user interaction:

```
# pip install accelerate transformers
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load model directly from Hugging Face Hub
tokenizer = AutoTokenizer.from_pretrained("google/txgemma-27b-chat")
model = AutoModelForCausalLM.from_pretrained(
    "google/txgemma-27b-chat",
    device_map="auto",
)

# Formatted TDC prompt (see "Formatting prompts for therapeutic tasks" section above)
prompt = TDC_PROMPT

# Format prompt in the conversational format
messages = [
    { "role": "user", "content": prompt}
]

# Apply the tokenizer's built-in chat template
chat_prompt = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_
```

At this point, the prompt contains the following text:

```
<bos><start_of_turn>user
Instructions: Answer the following question about drug properties.
Context: As a membrane separating circulating blood and brain extracellular fluid, th
Question: Given a drug SMILES string, predict whether it
(A) does not cross the BBB (B) crosses the BBB
Drug SMILES: CN1C(=O)CN=C(C2=CCCCC2)c2cc(Cl)ccc21
Answer:<end_of_turn>
<start_of_turn>model
```

As you can see, each turn is preceded by a **<start_of_turn>** delimiter and then the role of the entity (either **user**, for content supplied by the user, or **model**, for LLM responses). Turns finish with the **<end_of_turn>** token.

You can follow this format to build the prompt manually if you need to do it without the tokenizer's chat template.

After the prompt is ready, generation can be performed as follows:

```
inputs = tokenizer.encode(chat_prompt, add_special_tokens=False, return_tensors="pt")
outputs = model.generate(input_ids=inputs.to("cuda"), max_new_tokens=8)
response = tokenizer.decode(outputs[0, len(inputs[0]):], skip_special_tokens=True)
print(response)
```

For multiple interactions, append the model response and user prompt for an additional turn to the chat message history and perform generation in the same way:

```
messages.extend([
    { "role": "assistant", "content": response },
    { "role": "user", "content": "Explain your reasoning based on the molecule struct
])

chat_prompt = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_
inputs = tokenizer.encode(chat_prompt, add_special_tokens=False, return_tensors="pt")
outputs = model.generate(input_ids=inputs.to("cuda"), max_new_tokens=512)
response = tokenizer.decode(outputs[0, len(inputs[0]):], skip_special_tokens=True)
print(response)
```

Alternatively, you can use the **`pipeline`** API, which abstracts away details such as applying the template using the tokenizer:


```
# pip install transformers
from transformers import pipeline

# Instantiate a text generation pipeline using the model
pipe = pipeline(
    "text-generation",
    model="google/txgemma-27b-chat",
    device="cuda",
)

# Formatted TDC prompt (see "Formatting prompts for therapeutic tasks" section above)
prompt = TDC_PROMPT

# Format prompt in the conversational format for initial turn
messages = [
    { "role": "user", "content": prompt}
```

```
]

# Generate response for initial turn
outputs = pipe(messages, max_new_tokens=8)
print(outputs[0]["generated_text"][-1]["content"].strip())

# Append user prompt for an additional turn
messages = outputs[0]["generated_text"]
messages.append(
    { "role": "user", "content": "Explain your reasoning based on the molecule struct
)

# Generate response for additional turn
outputs = pipe(messages, max_new_tokens=512)
print(outputs[0]["generated_text"][-1]["content"].strip())
```

Examples

See the following Colab notebooks for examples of how to use TxGemma:

- To give the model a quick try, running it locally with weights from Hugging Face, see Quick start notebook in Colab (https://colab.research.google.com/github/google-gemini/gemma-cookbook/blob/main/TxGemma/%5BTxGemma%5DQuickstart_with_Hugging_Face.ipynb), which includes some example eval tasks from TDC.
- For a demo of how to fine-tune TxGemma in Hugging Face, see our Fine-tuning notebook i Colab (https://colab.research.google.com/github/google-gemini/gemma-cookbook/blob/main/TxGemma/%5BTxGemma%5DFinetune_with_Hugging_Face.ipynb).
- For a demo of how TxGemma can be used as a tool as part of a larger agentic workflow powered by Gemini 2 see the Agentic workflow notebook in Colab (https://colab.research.google.com/github/google-gemini/gemma-cookbook/blob/main/TxGemma/%5BTxGemma%5DAgentic_Demo_with_Hugging_Face.ipynb).

Model architecture overview

- TxGemma is based on the Gemma 2 family of lightweight, state-of-the-art open LLMs. It utilizes a decoder-only transformer architecture.

- Base Model: Gemma 2 (2B, 9B, and 27B parameter versions).
- Fine-tuning Data: Therapeutics Data Commons, a collection of instruction-tuning datasets covering diverse therapeutic modalities and targets.
- Training Approach: Instruction fine-tuning using a mixture of therapeutic data (TxT) and, for conversational variants, general instruction-tuning data.
- Conversational Variants: TxGemma-Chat models (9B and 27B) are trained with a mixture of therapeutic and general instruction-tuning data to maintain conversational abilities.

Technical Specifications

- Model type: Decoder-only Transformer (based on Gemma 2)
- Key publication: TxGemma: Efficient and Agentic LLMs for Therapeutics (https://arxiv.org/abs/2504.06196)
- Model created: 2025-03-18 (From the TxGemma Variant Proposal)
- Model Version: 1.0.0

Performance & Validation

TxGemma's performance has been validated on a comprehensive benchmark (https://tdcommons.ai/) of 66 therapeutic tasks derived from TDC.

Key performance metrics

- Aggregated Improvement: Improves over the original Tx-LLM paper (https://arxiv.org/abs/2406.06316) on 45 out of 66 therapeutic tasks.
- Best-in-Class Performance: Surpasses or matches best-in-class performance on 50 out of 66 tasks, exceeding specialist models on 26 tasks. See Table A.11 (https://arxiv.org/abs/2504.06196) of the TxGemma paper for the full breakdown.

Inputs and outputs

- **Input:** Text. For best performance, text prompts should be formatted according to the TDC structure, including instructions, context, question, and, optionally, few-shot examples. Inputs can include SMILES strings, amino acid sequences, nucleotide sequences, and natural language text.

- **Output:** Text.

Dataset details

Training dataset

Therapeutics Data Commons: A curated collection of instruction-tuning datasets covering 66 tasks spanning the discovery and development of safe and effective medicine. This includes over 15 million data points across different biomedical entities. Released TxGemma models are only trained on datasets with commercial licenses, whereas models in our publication are also trained on datasets with non-commercial licenses.

General Instruction-Tuning Data: Used for TxGemma-Chat in combination with TDC.

Evaluation dataset

Therapeutics Data Commons: The same 66 tasks used for training are used for evaluation, following TDC's recommended methodologies for data splits (random, scaffold, cold-start, combination, and temporal).

License

The use of TxGemma is governed by the Health AI Developer Foundations terms of use (https://developers.google.com/health-ai-developer-foundations/terms).

Implementation information

This section contains details about the model internals.

Software

Training was done using JAX (https://github.com/jax-ml/jax).

JAX allows researchers to take advantage of the latest generation of hardware, including TPUs, for faster and more efficient training of large models.

Use and limitations

Intended use

- Research and development of therapeutics.

Benefits

TxGemma provides a versatile and powerful tool for accelerating therapeutic development. It offers:

- Strong performance across a wide range of tasks.
- Data efficiency compared to larger models.
- A foundation for further fine-tuning from private data.
- Integration into agentic workflows.

Limitations

- Trained on public data from TDC.
- Task-specific validation remains an important aspect of downstream model development by the end user.
- As with any research, developers should ensure that any downstream application is valida to understand performance using data that is appropriately representative of the intended setting for the specific application (e.g., age, sex, gender, condition, scanner, etc.).

Citation

```
@article{wang2025txgemma,
    title={TxGemma: Efficient and Agentic LLMs for Therapeutics},
    author={Wang, Eric and Schmidgall, Samuel and Jaeger, Paul F. and Zhang, Fan and
    year={2025},
}
```

Find the paper here (https://arxiv.org/abs/2504.06196).

Last updated 2025-04-10 UTC.