SmolLM

Created by	Ⓢ Sourav das
Created time	@June 19, 2025 4:06 AM
Category	LLM
Last updated time	@June 19, 2025 6:07 AM

SmolLM and SmolLM2: Compact and Efficient Language Models by Hugging Face

Introduction

SmolLM introduced a family of small models trained on high-quality datasets to compete with larger models in specific tasks. **SmolLM2** builds on this foundation, enhancing architecture, training strategies, and instruction-following capabilities. This blog provides a comprehensive overview of both model families, incorporating tables to summarize key details on architecture, training, features, efficiency, and limitations.

Detailed Architecture

Both SmolLM and SmolLM2 utilize a **transformer decoder architecture**, optimized for efficiency and scalability. Below, we outline the architectural details for each model size, highlighting differences between SmolLM and SmolLM2, with a table summarizing key configurations.

Common Architectural Features

- **Transformer Decoder**: Decoder-only architecture for autoregressive tasks like text generation and instruction following.
- **Context Length**: 2048 tokens, extendable to 8k tokens via long-context fine-tuning.
- **Embedding Tying**: Shared input and output embedding matrices to reduce memory usage.
- **Tokenizer**: Custom tokenizer with a 49,152-token vocabulary, trained on the SmolLM-Corpus.
- **Precision**: Models use **bfloat16**, with 8-bit quantization options for reduced memory footprint.

SmolLM Architecture

The SmolLM family (135M, 360M, and 1.7B parameters) prioritizes efficiency, especially for smaller models.

135M and 360M Parameter Models

- **Design Philosophy**: Inspired by MobileLLM, these models emphasize **depth over width** to balance performance and computational efficiency, ideal for devices like smartphones.
- **Grouped-Query Attention (GQA)**: Reduces memory and computation by grouping query heads to share key and value vectors, maintaining performance for on-device use.

1.7B Parameter Model

- **Traditional Architecture**: Adopts a LLaMA2-inspired design with fewer efficiency-focused optimizations like GQA, leveraging larger capacity for complex tasks.

Architecture Comparison Table

Feature	SmolLM-135M	SmolLM-360M	SmolLM-1.7B	SmolLM2-135M	SmolLM2-360M	SmolLM2-1.7B
Parameters	135M	360M	1.7B	135M	360M	1.7B
Attention Mechanism	GQA	GQA	Multi-Head	GQA	GQA	Multi-Head (32 heads, 32 KV)
Layers	Not specified	Not specified	Not specified	Not specified	Not specified	24
Embedding Length	Not specified	Not specified	Not specified	Not specified	Not specified	2048
Feed-Forward Length	Not specified	Not specified	Not specified	Not specified	Not specified	8192
Context Length	2048 (8k ext.)	2048 (8k ext.)	2048 (8k ext.)	2048 (8k ext.)	2048 (8k ext.)	2048 (8k ext.)
Position Embedding	RoPE	RoPE	RoPE	RoPE	RoPE	RoPE (base 10k, dim 64)

Key Differences

- **SmolLM2 Improvements**: Enhanced attention mechanisms and layer configurations, especially in the 1.7B model, with a focus on instruction-following.
- **Instruction Tuning**: SmolLM2 emphasizes tasks like text rewriting, summarization, and function calling.

Training and Optimization

SmolLM and SmolLM2 are trained on high-quality datasets with advanced optimization techniques to maximize performance and efficiency. The following table summarizes training details.

Training Data

Both model families use the **SmolLM-Corpus**:

- **FineWeb-Edu**: 220B tokens of educational web content.
- **Cosmopedia v2**: 28B tokens of synthetic textbooks and stories.
- **Python-Edu**: 4B tokens of educational Python samples.
- **SmolLM2 Additions**: FineMath, Stack-Edu, and SmolTalk for enhanced math, coding, and conversational skills.

Training Details Table

Model	Parameters	Training Tokens	Training Stages	Hardware	Learning Rate Scheduler
SmolLM-135M	135M	2T	Single-stage	64 H100 GPUs	WSD (20% decay)
SmolLM-360M	360M	4T	Single-stage	64 H100 GPUs	WSD (20% decay)
SmolLM-1.7B	1.7B	11T	Multi-stage	256 H100 GPUs	WSD (20% decay)
SmolLM2-135M	135M	2T	Single-stage	64 H100 GPUs	WSD (20% decay)

Model	Parameters	Training Tokens	Training Stages	Hardware	Learning Rate Scheduler
SmolLM2-360M	360M	4T	Single-stage	64 H100 GPUs	WSD (20% decay)
SmolLM2-1.7B	1.7B	11T	Multi-stage	256 H100 GPUs	WSD (20% decay)

Optimization Techniques

- **Training Framework**: **Nanotron** for high-performance distributed training.
- **Precision**: **bfloat16** with 8-bit quantization options.
- **Instruction Tuning**:
  - **Supervised Fine-Tuning (SFT)**: Aligns models with human-like responses using curated datasets.
  - **Direct Preference Optimization (DPO)**: Uses UltraFeedback to enhance instruction-following, particularly in SmolLM2-1.7 B.
- **Learning Rate**: Warmup Stable Decay (WSD) scheduler ensures stable convergence.

Key Features and Efficiency

SmolLM and SmolLM2 are designed for efficiency and versatility, making them ideal for on-device NLP applications. The following table summarizes memory usage and efficiency.

Memory Usage and Efficiency Table

Model	Parameters	Memory (bfloat16)	Memory (8-bit)	Key Applications
SmolLM-135M	135M	269.03 MB	162.87 MB	Text generation, lightweight tasks
SmolLM-360M	360M	723.56 MB	Not specified	Text generation, coding, education
SmolLM-1.7B	1.7B	Not specified	1812.14 MB	Complex tasks, instruction following
SmolLM2-135M	135M	269.03 MB	162.87 MB	Text generation, lightweight tasks
SmolLM2-360M	360M	723.56 MB	Not specified	Text generation, coding, education
SmolLM2-1.7B	1.7B	Not specified	1812.14 MB	Instruction following, function calling

Key Features

- **Compact Design**: Sizes cater to varying computational budgets.
- **On-Device Deployment**: Low memory footprints enable use on smartphones, tablets, and embedded systems.
- **Applications**:
  - Text generation, summarization, and rewriting.
  - Coding (Python) and educational content generation.
  - Function calling (SmolLM2-1.7B).
- **Extended Context**: Up to 8k tokens via RoPE scaling.
- **Performance**: Outperforms models like MobileLM-125M and Qwen2.5-1.5B in benchmarks for reasoning, knowledge, and coding.

Efficiency

- **Low Latency**: Optimized for real-time applications.
- **Energy Efficiency**: Reduced power consumption for battery-powered devices.
- **Quantization**: 8-bit options minimize resource demands.

Practical Usage

Example for SmolLM2-1.7B with Hugging Face `transformers`:

```
from transformers import AutoModelForCausalLM, AutoTokenizer
checkpoint = "HuggingFaceTB/SmolLM2-1.7B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = AutoModelForCausalLM.from_pretrained(checkpoint).to("cuda")
inputs = tokenizer.encode("Explain quantum mechanics", return_tensors="pt").to("cuda")
outputs = model.generate(inputs, max_new_tokens=100)
print(tokenizer.decode(outputs[0]))import json
import re
from typing import Optional

from jinja2 import Template
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
from transformers.utils import get_json_schema



system_prompt = Template("""You are an expert in composing functions. You are given a question and a s
et of possible functions.
Based on the question, you will need to make one or more function/tool calls to achieve the purpose.
If none of the functions can be used, point it out and refuse to answer.
If the given question lacks the parameters required by the function, also point it out.

You have access to the following tools:
<tools>{{ tools }}</tools>

The output MUST strictly adhere to the following format, and NO other text MUST be included.
The example format is as follows. Please make sure the parameter type is correct. If no function call is nee
ded, please make the tool calls an empty list '[]'.
<tool_call>[
{"name": "func_name1", "arguments": {"argument1": "value1", "argument2": "value2"}},
... (more tool calls as required)
]</tool_call>""")



def prepare_messages(
    query: str,
    tools: Optional[dict[str, any]] = None,
    history: Optional[list[dict[str, str]]] = None
) → list[dict[str, str]]:
    """Prepare the system and user messages for the given query and tools.
    Args:
        query: The query to be answered.
        tools: The tools available to the user. Defaults to None, in which case if a
            list without content will be passed to the model.
        history: Exchange of messages, including the system_prompt from
            the first query. Defaults to None, the first message in a conversation.
    """
    if tools is None:
        tools = []
    if history:
        messages = history.copy()
        messages.append({"role": "user", "content": query})
    else:
        messages = [
            {"role": "system", "content": system_prompt.render(tools=json.dumps(tools))},
            {"role": "user", "content": query}
        ]
    return messages


def parse_response(text: str) → str | dict[str, any]:
    """Parses a response from the model, returning either the
    parsed list with the tool calls parsed, or the
    model thought or response if couldn't generate one.

    Args:
        text: Response from the model.
    """
    pattern = r"<tool_call>(.*?)</tool_call>"
    matches = re.findall(pattern, text, re.DOTALL)
    if matches:
        return json.loads(matches[0])
    return text


model_name_smollm = "HuggingFaceTB/SmolLM2-1.7B-Instruct"
model = AutoModelForCausalLM.from_pretrained(model_name_smollm, device_map="auto", torch_dtype
="auto", trust_remote_code=True)
tokenizer = AutoTokenizer.from_pretrained(model_name_smollm)

from datetime import datetime
import random

def get_current_time() → str:
    """Returns the current time in 24-hour format.

    Returns:
        str: Current time in HH:MM:SS format.
    """
    return datetime.now().strftime("%H:%M:%S")


def get_random_number_between(min: int, max: int) -> int:
    """
    Gets a random number between min and max.

    Args:
        min: The minimum number.
        max: The maximum number.

    Returns:
        A random number between min and max.
    """
    return random.randint(min, max)


tools = [get_json_schema(get_random_number_between), get_json_schema(get_current_time)]

toolbox = {"get_random_number_between": get_random_number_between, "get_current_time": get_current
_time}

query = "Give me a number between 1 and 300"

messages = prepare_messages(query, tools=tools)

inputs = tokenizer.apply_chat_template(messages, add_generation_prompt=True, return_tensors="pt").to
(model.device)
outputs = model.generate(inputs, max_new_tokens=512, do_sample=False, num_return_sequences=1, eos_
token_id=tokenizer.eos_token_id)
result = tokenizer.decode(outputs[0][len(inputs[0]):], skip_special_tokens=True)

tool_calls = parse_response(result)
# [{'name': 'get_random_number_between', 'arguments': {'min': 1, 'max': 300}}

# Get tool responses
tool_responses = [toolbox.get(tc["name"])(*tc["arguments"].values()) for tc in tool_calls]
# [63]

# For the second turn, rebuild the history of messages:
history = messages.copy()
# Add the "parsed response"
history.append({"role": "assistant", "content": result})
query = "Can you give me the hour?"
history.append({"role": "user", "content": query})
```

```
inputs = tokenizer.apply_chat_template(history, add_generation_prompt=True, return_tensors="pt").to(model.device)
outputs = model.generate(inputs, max_new_tokens=512, do_sample=False, num_return_sequences=1, eos_token_id=tokenizer.eos_token_id)
result = tokenizer.decode(outputs[0][len(inputs[0]):], skip_special_tokens=True)

tool_calls = parse_response(result)
tool_responses = [toolbox.get(tc["name"])(*tc["arguments"].values()) for tc in tool_calls]
# ['07:57:25']
```

Limitations

Despite their strengths, SmolLM and SmolLM2 have limitations, summarized below:

Limitations Table

Limitation	**Description**
Language Scope	Primarily English-focused, with limited performance in non-English languages.
Accuracy and Bias	Potential inaccuracies or biases in generated content; requires verification.
Task Complexity	May underperform larger models in complex reasoning or specialized domains.
Resource Constraints	1.7B models may be challenging for very low-end devices.
Function Calling	Limited to SmolLM2-1.7B, less robust than larger models.
Training Data Dependence	Performance tied to SmolLM-Corpus, which may miss some edge cases or domains.