WHAT IS GENERATIVE AI CAPABLE OF?

Text Generation

Prompt: Write a proof of the fact that there are infinitely many primes; do it in the style of a Shakespeare play through a dialogue between two parties arguing over the proof.

GPT-4: Enter FIRST PARTY and SECOND PARTY.

FIRST PARTY: Thou dost not understand, my friend, There simply is no upper end To primes, those numbers indivisible, No matter if it seems impossible.

SECOND PARTY: But how dost thou so surely know? To prove this truth, what canst thou show?

FIRST PARTY: Consider this, my doubtful peer, A clever proof that will make clear: Assume that there's a final prime, The largest one we'll see in time. Let's call this number "P" for short, And then, my friend, we will purport To build a number "Q" that's new, One larger than "P" and prime too.

SECOND PARTY: But how, my friend, will this be done? The proof is not yet fairly won.

FIRST PARTY: It's simple, friend, just wait and see, We'll multiply the primes, that's key. All primes from two to "P" combined, Will make this new "Q" we will find. But one thing more we must include, To make the proof more watertight, We'll add a one onto the end, And now the proof is in our sight.

SECOND PARTY: I see it now, it's crystal clear, For "Q" must be a prime, my dear. It can't divide by any prime, All smaller than itself in time. And since we chose "P" as the last, This proves that our assumption's past. There is no final prime, we see, The numbers go on endlessly.

Example from https://arxiv.org/pdf/2303.12712.pdf

Image Editing

A variety of tasks involve automatic editing of an image:

- **Inpainting** fills in the (pre-specified) missing pixels
- **Colorization** restores color to a greyscale image
- **Uncropping** creates a photo-realistic reconstruction of a missing side of an image

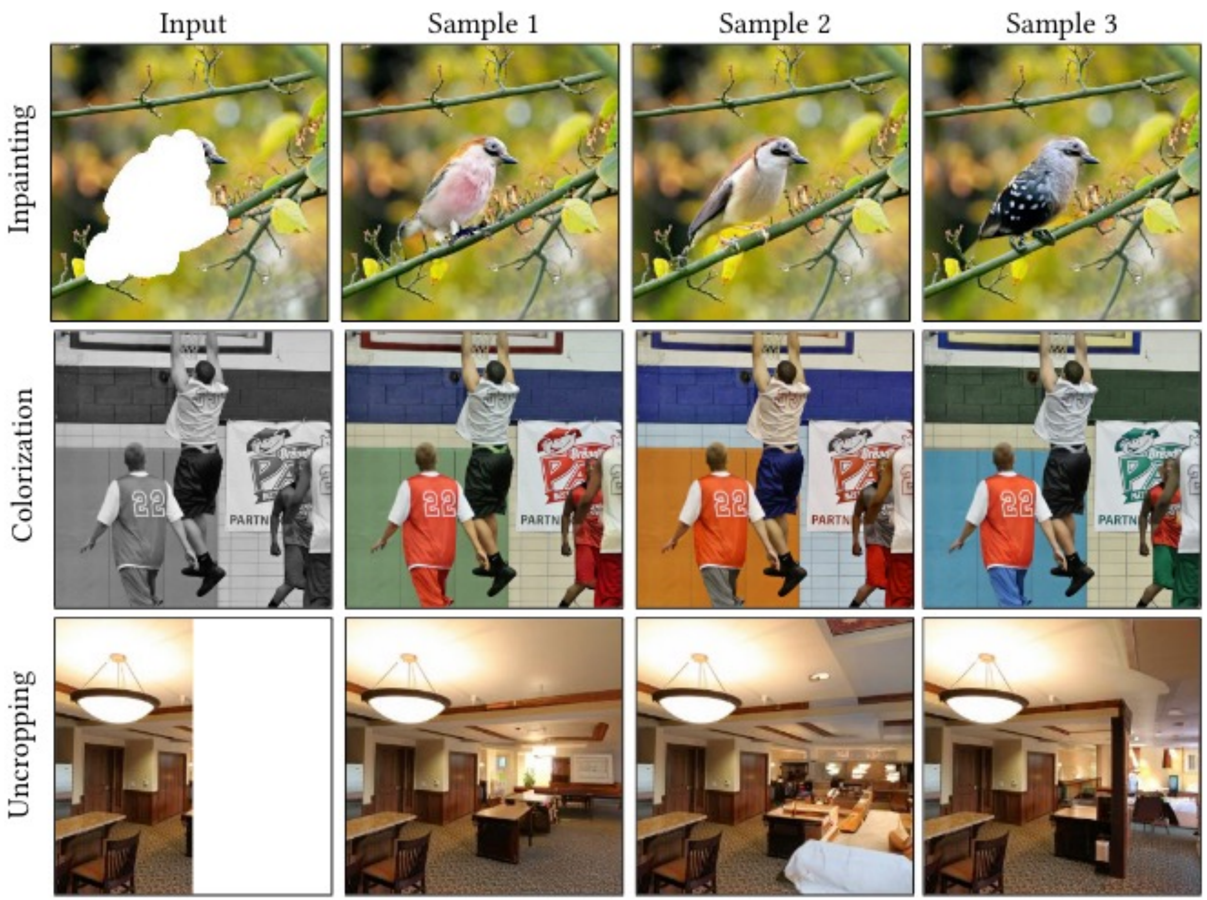


Figure from Saharia et al. (2022)

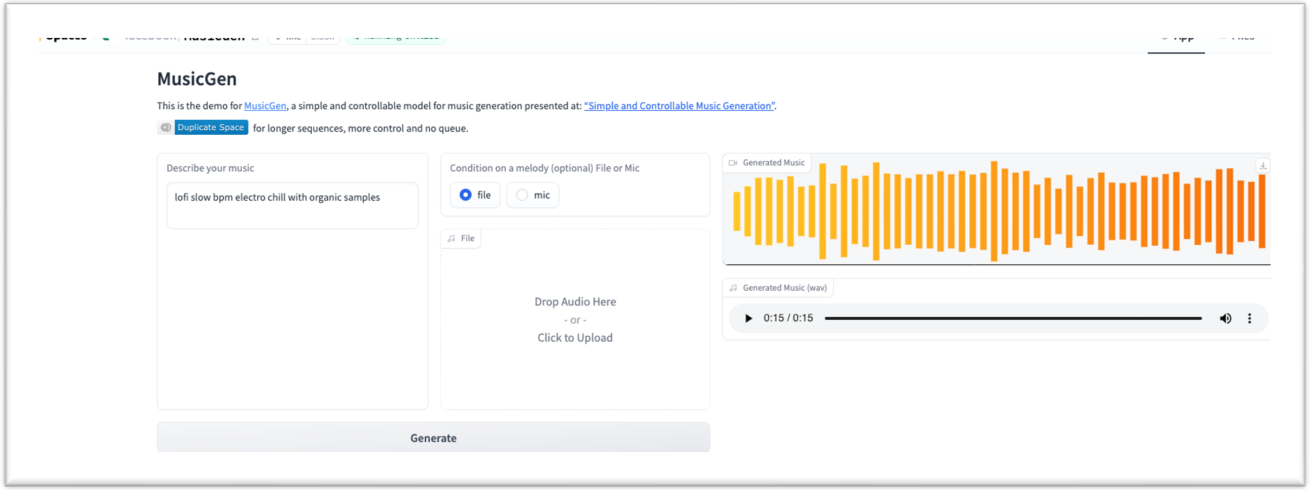
Text-to-Image Generation

- Given a text description, sample an image that depicts the prompt
- The following images are samples from SDXL with refinement

Prompt: close up headshot, futuristic **old man**, wild hair sly smile in front of gigantic UFO, dslr, sharp focus, dynamic composition, **rule of thirds**

Figure from https://stablediffusionweb.com/

Music Generation



MusicGen

- A transformer decoder model over quantized units (discrete elements of a codebook of audio frames)
- Interleaves sounds by adjusting how codebooks attend to each other
- Permits conditioning on text and/or audio samples

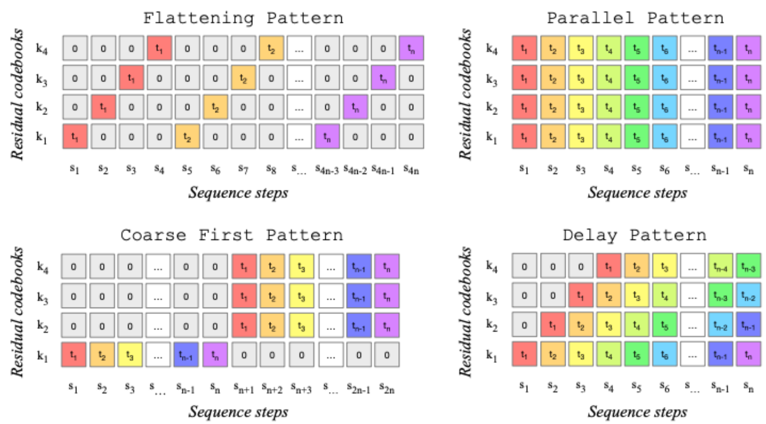


Figure from https://arxiv.org/pdf/2306.05284.pdf

Code Generation

Figure 1.3: We queried GPT-4 three times, at roughly equal time intervals over the span of a month while the system was being refined, with the prompt “Draw a unicorn in TikZ”. We can see a clear evolution in the sophistication of GPT-4’s drawings.

GPT-4 vs ChatGPT

Prompt:

```
A function f is called {increasing}_{emphasize!} if f(s) <= f(t) holds for every s <= t. Consider the formula:
limit_{x goes to 0} {limit when n approaches infinite} underscore{
\matrix[ [ sum of i from 1 to n (f(nx) - integral from 0 to x sin(nx) dx ),
((x + 2/8)^2 - 5)/((4x-1/x^2) /(n xor 3)), \\ , ..., partial f(x/n)
over partial x * (e^{-i*omega n x} + 3(x#7) ) , f inverse of (x mod n) ]]
multiply \matrix{1, 1, ..., 1 \\ 1, 1, ..., 1} }
{grey_color{this gives us a good impression about the function}}

Convert the above paragraph to LaTeX.
```

GPT-4:

A function f is called *increasing* if $f(s) \leq f(t)$ holds for every $s \leq t$. Consider the formula:

$$\lim_{x\to 0}\lim_{n\to\infty}\underbrace{\begin{bmatrix}\sum_{i=1}^{n}\left(f(nx)-\int_0^x \sin(nx)\,dx\right), & \frac{(x+\frac{2}{8})^2-5}{\frac{4x-1/x^2}{n\oplus 3}}, \\ \vdots & \vdots \\ \frac{\partial f(x/n)}{\partial x}\cdot\left(e^{-i\omega n x}+3(x\#7)\right), & f^{-1}(x \bmod n)\end{bmatrix}\times\begin{bmatrix}1 & 1 & \cdots & 1\\ 1 & 1 & \cdots & 1\end{bmatrix}}_{\text{this gives us a good impression about the function}}$$

GPT-4 vs ChatGPT

Prompt:

```
I want to write a customized optimizer in pytorch, the optimizer should do:
1. the parameters for the optimizer is learning rate, momentum ,weigth decay, d_dim, k and factor alpha
2. the optimizer goes through each parameter in the network, and then
-- Get the gradient of this parameter, reshape it to 2D by looping through the dimensions, and keeping the dimension whose actual
dimension (shape) is equal to d_dim, do an SVD decomposition of the parameter so W = U\Sigma V.
-- Keep the top k singular vectors of W, so we get a new matrix W_k = U \Sigma_k V.
-- Keep the top 2k singular vectors of W, so we get a new matrix W_{2k} = U \Sigma_{2k} V.
-- Normalize W_k according to the momentum of the F-norm of W_{2k}.
-- Truncate each coordinate of W_k to the top alpha  percentile of all the coordinates, in absolute value.
-- Apply momentum on W_k, and then update the network parameters using this momentum.
-- Apply weight decay.
```

GPT-4:

```python
def step(self):
    for group in self.param_groups:
        for p in group['params']:
            if p.grad is None:
                continue
            grad = p.grad.data
            # reshape grad to 2D
            grad_2d = grad.reshape(-1, group['d_dim'])
            # SVD decomposition
            U, S, V = torch.svd(grad_2d)
            # keep top k singular vectors
            G_k = U[:, :group['k']] @ torch.diag(S[:group['k']]) @
            V[:, :group['k']].T
            # keep top 2k singular vectors
            G_2k = U[:, :2*group['k']] @ torch.diag(S[:2*group['k']])
            @ V[:, :2*group['k']].T
            # normalize G_k
            G_k /= torch.norm(G_2k, p=2)
            # truncate G_k
            threshold = torch.quantile(torch.abs(G_k),
            group['alpha'])
            G_k[torch.abs(G_k) < threshold] = 0
            # apply momentum
            if 'momentum_buffer' not in group:
                group['momentum_buffer'] = torch.zeros_like(G_k)
            group['momentum_buffer'] = group['momentum'] *
            group['momentum_buffer'] + G_k
            # update parameters
            p.data.add_(-group['lr'] * group['momentum_buffer'])
            # apply weight decay
            if group['weight_decay'] != 0:
                p.data.add_(-group['weight_decay'] * group['lr'],
                p.data)
```

ChatGPT:

```python
def step(self):
    for group in self.param_groups:
        for p in group['params']:
            if p.grad is None:
                continue
            grad = p.grad.data
            grad = grad.reshape(-1, grad.shape[-1]) # reshape to 2D

            # SVD decomposition
            u, s, v = torch.svd(grad)

            # Keep top k and 2k singular vectors
            k = group['k']
            u_k = u[:,:k]
            s_k = s[:k]
            v_k = v[:,:k]
            u_2k = u[:,:2*k]
            s_2k = s[:2*k]
            v_2k = v[:,:2*k]

            # Normalize according to momentum of F-norm of G_2k
            grad_k = u_k @ torch.diag(s_k) @ v_k.t()
            norm_2k = s_2k.norm()
            grad_k = grad_k * group['momentum'] / norm_2k

            # Truncate to top alpha percentile
            alpha = group['alpha']
            percentile = grad_k.abs().flatten().kthvalue(int(alpha *
            grad_k.numel()))
            grad_k[grad_k.abs() < percentile] = 0

            # Apply weight decay
            p.data.add_(-group['lr'], grad_k + group['weight_decay']
            * p.data)
```

Figure 3.4: GPT-4 vs ChatGPT at implementing a customized Pytorch optimizer module. We highlight the astuteness of GPT-4 vs the mistake by ChatGPT at ‘applying momentum’.

Example from https://arxiv.org/pdf/2303.12712.pdf

Video Generation

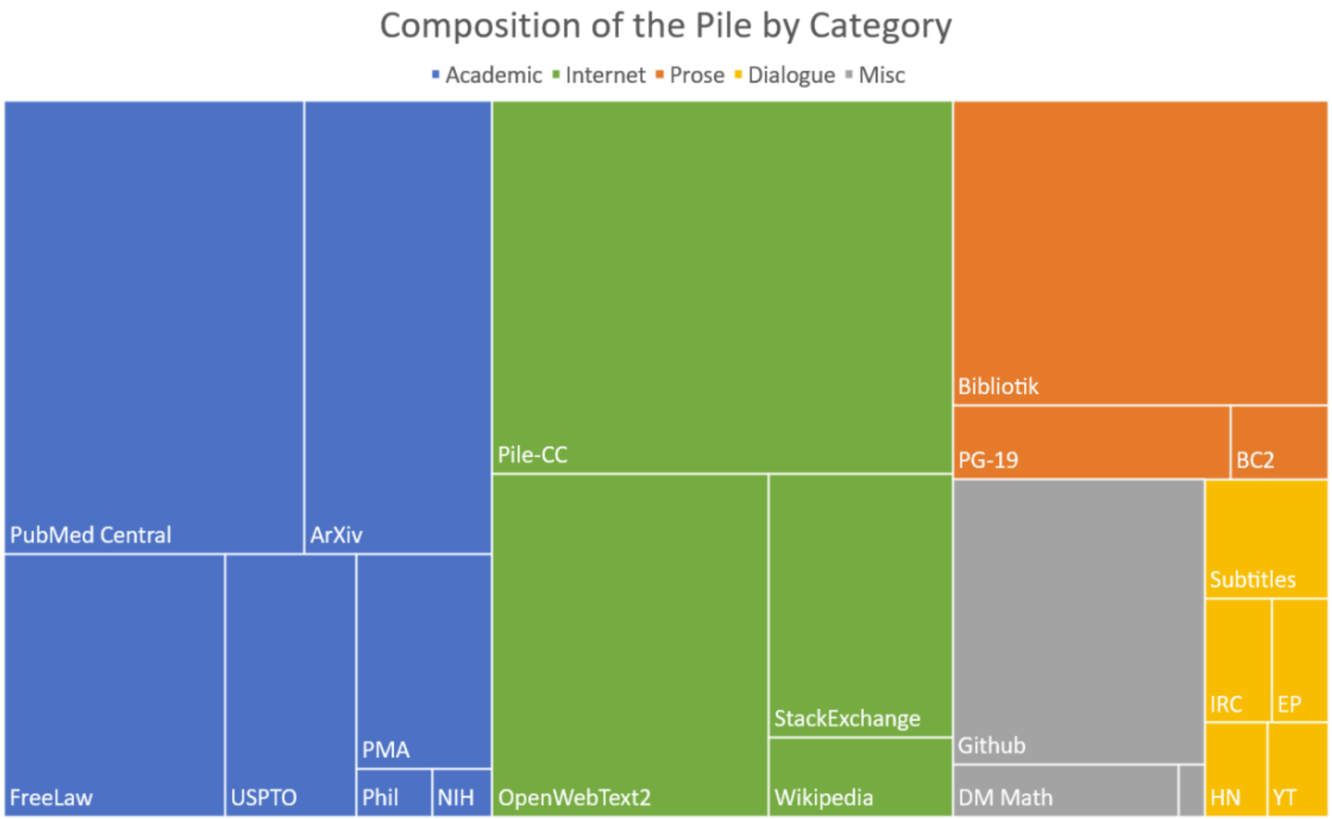
- Latent diffusion models use a low-dimensional latent space for efficiency
- **Key question:** how to generate multiple correlated frames?
- 'Align your latents' inserts temporal convolution / attention between each spatial convolution / attention
- 'Preserve Your Own Correlation' includes temporally correlated noise

Figure from https://huggingface.co/stabilityai/stable-video-diffusion-img2vid-xt

SCALING UP

Training Data for LLMs

The Pile:
- An open source dataset for training language models
- Comprised of 22 smaller datasets
- Favors high quality text
- 825 Gb ≈ 1.2 trillion tokens



RLHF

- **InstructGPT** uses Reinforcement Learning with Human Feedback (RLHF) to **fine-tune** a **pre-trained** GPT model
- From the paper: "In human evaluations on our prompt distribution, outputs from the 1.3B parameter InstructGPT model are preferred to outputs from the 175B GPT-3, despite having 100x fewer parameters."

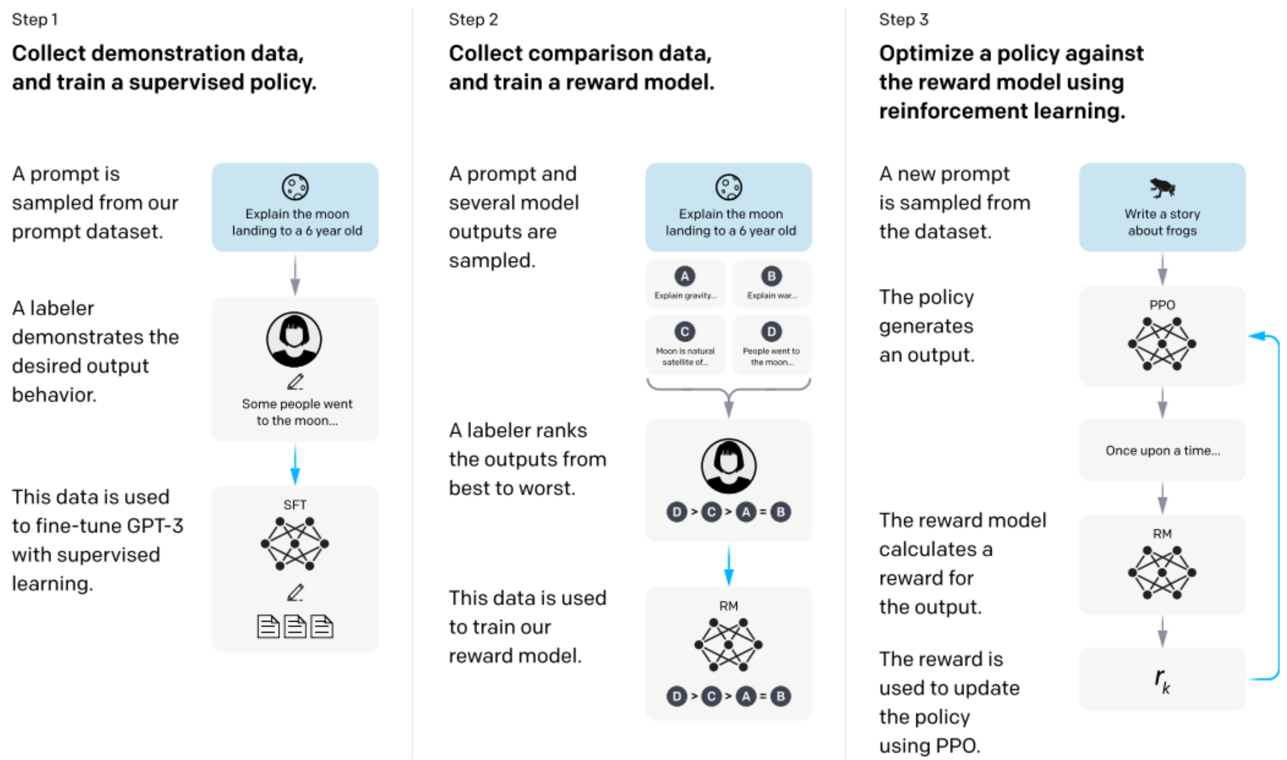


Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

Figure from https://arxiv.org/pdf/2203.02155.pdf

Memory Usage of LLMs

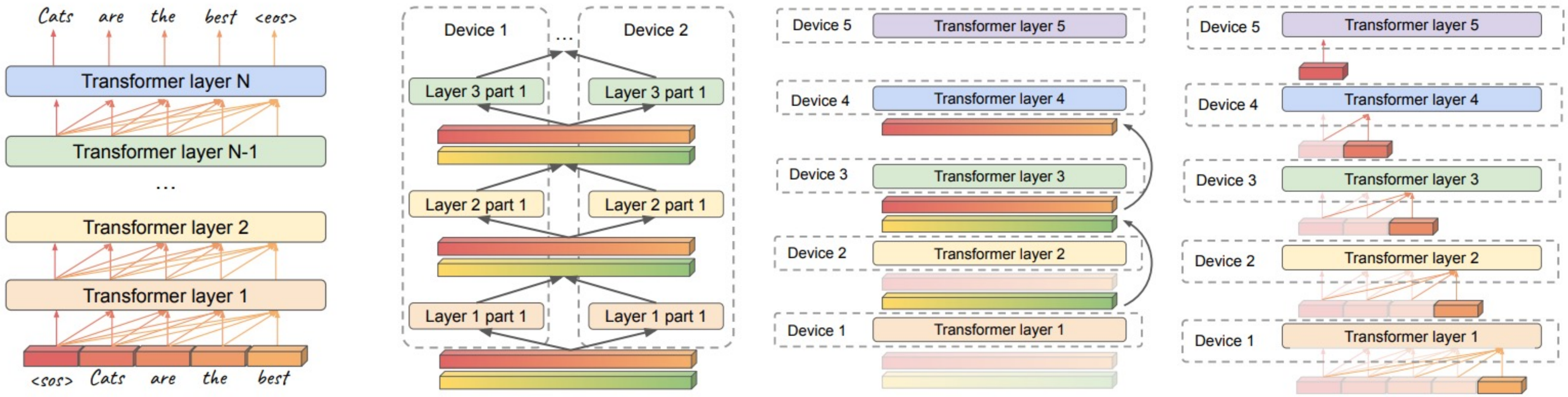
How to store a large language model in memory?

- **full precision:** 32-bit floats
- **half precision:** 16-bit floats
- Using half precision not only **reduces memory,** it also **speeds up** GPU computation
- *"Peak float16 matrix multiplication and convolution performance is 16x faster than peak float32 performance on A100 GPUs."* [from Pytorch docs]

Model	Megatron-LM	GPT-3
# parameters	8.3 billion	175 billion
full precision	30 Gb	651 Gb
half precision	15 Gb	325 Gb

GPU / TPU	Max Memory
TPU v2	16 Gb
TPU v3/v4	32 Gb
Tesla V100 GPU	32 Gb
NVIDIA RTX A6000	48 Gb
Tesla A100 GPU	80 Gb

Distributed Training: Model Parallel



(a) Transformer-based LM

(b) Operation partitioning (Megatron-LM)

(c) Microbatch-based pipeline parallelism (GPipe)

(d) Token-based pipeline parallelism (TeraPipe)

There are a variety of different options for how to distribute the model computation / parameters across multiple devices.

Matrix multiplication comprises most Transformer LM computation and can be divided along rows/columns of the respective matrices.

The most natural division is by layer: each device computes a subset of the layers, only that device stores the parameters and computation graph for those layers.

A more efficient solution is to divide computation by token *and* layer. This requires careful division of work and is specific to the Transformer LM.

Figure from https://arxiv.org/pdf/2102.07988.pdf

Cost to train

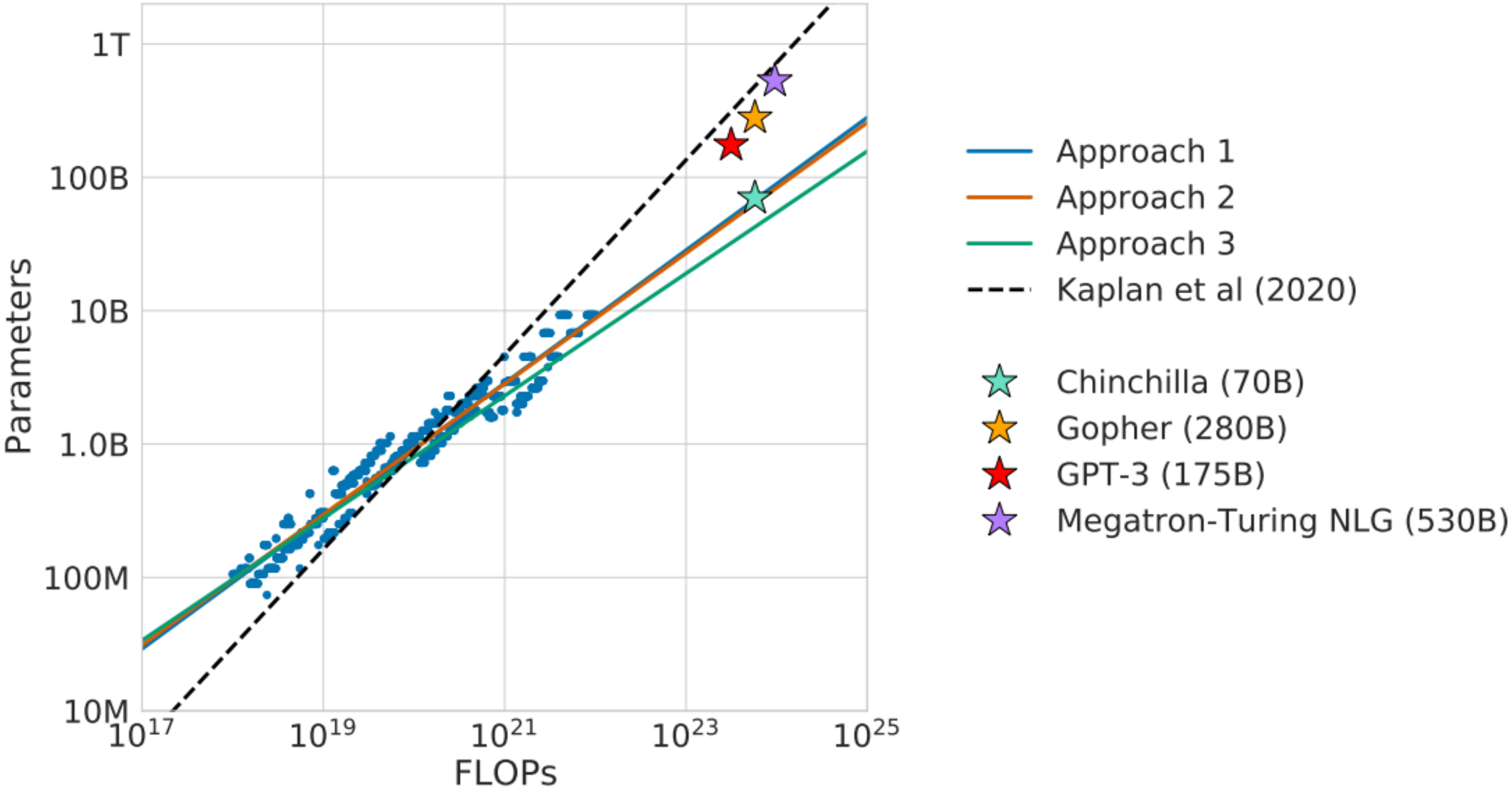
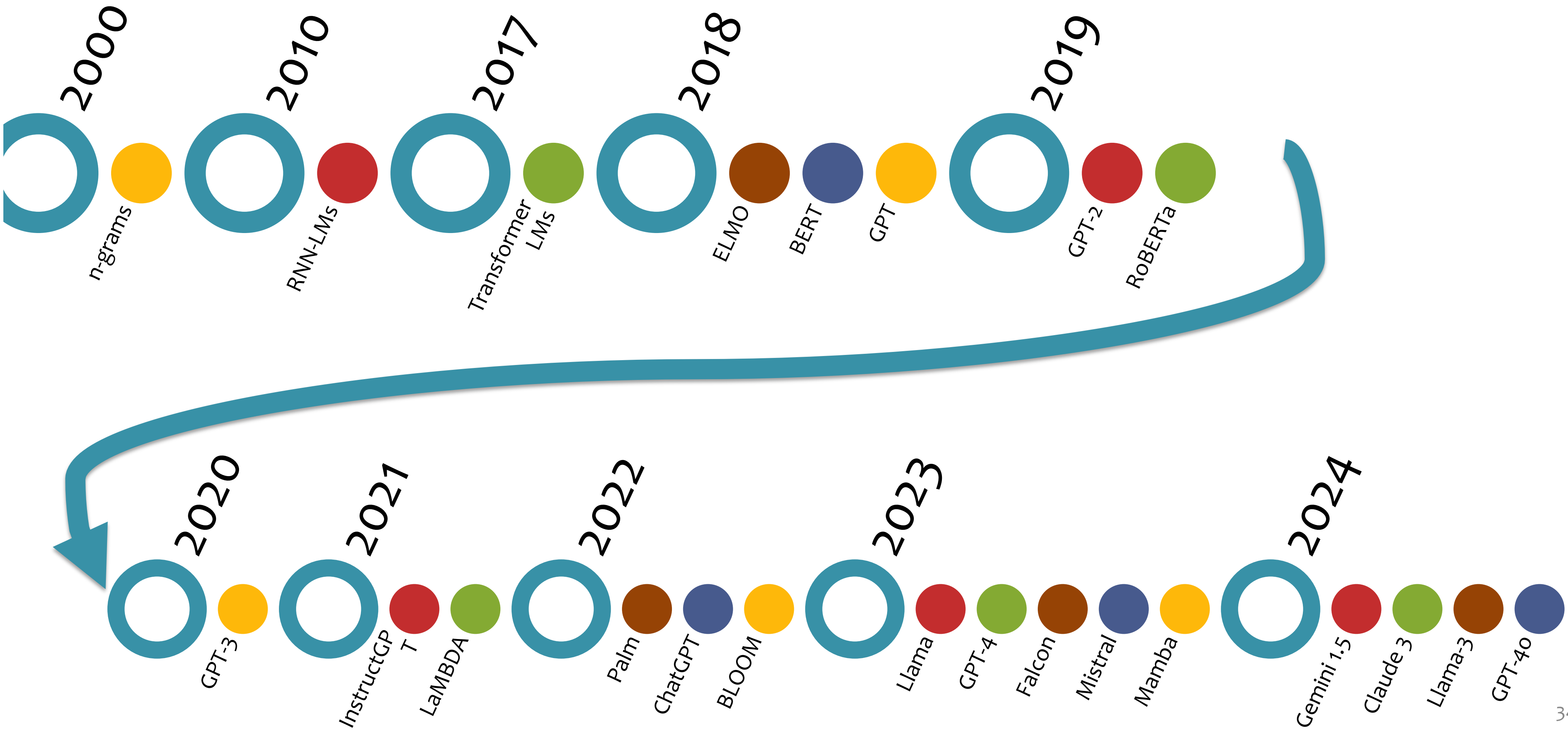
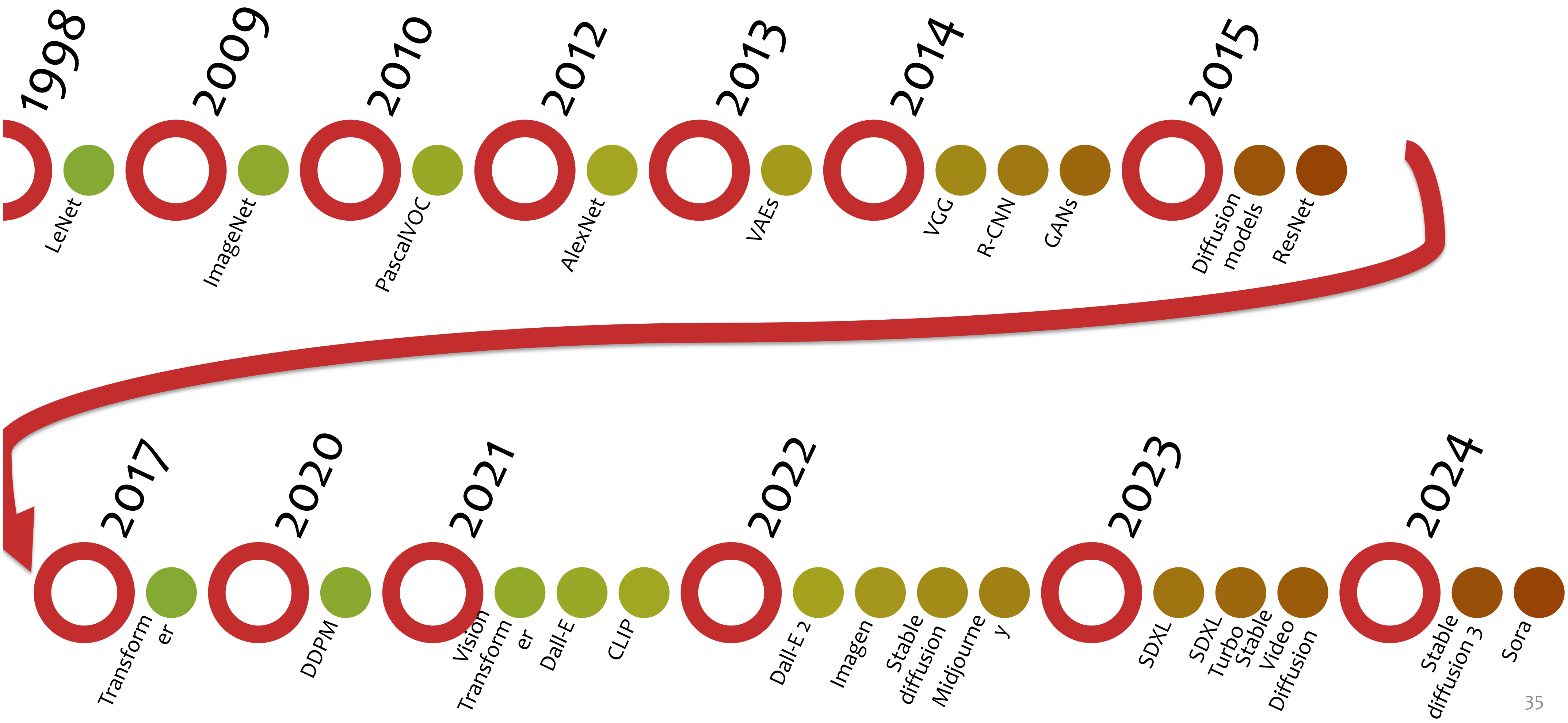


Figure from https://arxiv.org/pdf/2203.15556.pdf

Timeline: Language Modeling



Timeline: Image Generation



GENERATIVE AI IS PROBABILISTIC MODELING

GenAI is Probabilistic Modeling

$$p(x_{t+1} \mid x_1, \ldots, x_t)$$

What if I want to model EVERY possible interaction?

…or at least the interactions of the current variable with all those that came before it…

(RNN-LMs)

RNN Language Model

RNN Language Model: $p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p(w_t \mid f_{\boldsymbol{\theta}}(w_{t-1}, \ldots, w_1))$

$p(w_1, w_2, w_3, \ldots, w_6) =$

The						$p(w_1)$
The	bat					$p(w_2 \mid f_\theta(w_1))$
The	bat	made				$p(w_3 \mid f_\theta(w_2, w_1))$
The	bat	made	noise			$p(w_4 \mid f_\theta(w_3, w_2, w_1))$
The	bat	made	noise	at		$p(w_5 \mid f_\theta(w_4, w_3, w_2, w_1))$
The	bat	made	noise	at	night	$p(w_6 \mid f_\theta(w_5, w_4, w_3, w_2, w_1))$

Key Idea:

(1) convert all previous words to a **fixed length vector**

(2) define distribution $p(w_t \mid f_\theta(w_{t-1}, \ldots, w_1))$ that conditions on the vector