ChatGPT

Generative AI Self-Study Roadmap

1. Foundations: Programming & Mathematics

- **Python & ML Basics:** Comfort with Python (and libraries like PyTorch or TensorFlow) is essential [1]. Practice by coding simple ML models (regression, classification) or following PyTorch tutorials.
- **Mathematics:** Review linear algebra (vectors, matrices), calculus (derivatives, gradients), and probability/statistics (distributions, expectations). These underlie how models learn. Free resources like Khan Academy and 3Blue1Brown videos are helpful. Stanford's courses list these as prerequisites [1].
- **Core ML/DL Concepts:** Understand how neural networks work (forward/backward pass, activation functions, loss functions). If needed, take a free ML refresher (e.g. Andrew Ng's free online courses or MIT OpenCourseWare lectures). Knowing CNN basics is useful for image models.

2. Large Language Models (LLMs)

Large Language Models use the **Transformer** architecture to process and generate text [2]. They begin by **tokenizing** input text (using subword schemes like BPE or WordPiece) into numeric IDs and learning word embeddings plus positional encodings [2]. During generation, LLMs use decoding strategies (greedy/beam search or stochastic sampling such as top-*k* or nucleus sampling with a "temperature" parameter) to produce coherent text [3]. Pretraining involves huge text corpora and advanced optimizers (e.g. AdamW, learning-rate schedules) and produces the base model; fine-tuning adapts it to tasks or domains. Interactive resources – e.g. Hugging Face's free "Large Language Model Course" and visualizations (3Blue1Brown's attention video, Karpathy's nanoGPT tutorial) – build intuition for how attention and sampling work [4].

- **Tokenization & Embeddings:** Convert text to tokens (e.g. BPE, WordPiece) [2]. The model learns token embeddings and positional encodings to represent sequence order.
- **Transformer Architecture:** Stack of encoder/decoder (or decoder-only) layers with multi-head self-attention and feed-forward networks [2]. Understand how attention weights contextualize each token's representation.
- **Pretraining:** Typically autoregressive (GPT-style) or masked (BERT/T5) language modeling with cross-entropy loss. Trained on large web-scale corpora. No special citation needed here.
- **Decoding/Sampling:** At inference, use greedy or beam search, or sampling (top-*k*, nucleus) to trade off diversity vs. coherence [3]. Experiment with Hugging Face pipelines (e.g. `pipeline("text-generation")`).
- **Fine-Tuning & Prompting:** After pretraining, models can be fine-tuned on task-specific data, or used via prompt engineering (few-shot examples in the input). Study examples of both (Hugging Face docs have many).
- **Visualization & Understanding:** Use tools like BertViz or embedding projector to inspect attention. The Hugging Face LLM course points to 3Blue1Brown's visual intro and interactive LLM demos [4], which are free and clarify how tokens attend to each other.
- **Resources:** Free content includes the Hugging Face Transformers library (docs and example notebooks), and Stanford CS224N (NLP with Deep Learning) lectures/slides [5]. For hands-on

practice, try fine-tuning a small open model (e.g. GPT-2, GPT-Neo) on a public dataset (e.g. HF's `datasets`). Hugging Face's notebooks and EleutherAI's GitHub have many tutorials.

3. Image Generation

3.1 Autoencoders & Variational Autoencoders (VAEs)

Autoencoders are neural networks that **encode** an input (e.g. an image) into a low-dimensional code and **decode** it back to reconstruct the input. *Variational* Autoencoders introduce probability: the encoder outputs parameters (mean and variance) of a Gaussian latent distribution, and the decoder samples from this distribution to reconstruct [6]. The training loss is the **Evidence Lower Bound (ELBO)**: a reconstruction term plus a KL-divergence that forces the latent distribution toward a standard normal [7]. This encourages a smooth latent space – interpolating or sampling in latent space produces realistic new images. The figure above shows random digit images generated by a trained VAE on MNIST, illustrating how sampling the latent space yields new examples [6].

- **Architecture:** Convolutional encoder compresses the image into latent parameters (μ,σ); the decoder upsamples from a latent sample z back to the image.
- **Loss (ELBO):** $\mathcal{L} = \mathbb{E}_{q(z|x)}[-\log p(x|z)] + \text{KL}(q(z|x)|\ p(z))$, combining reconstruction loss and a regularizer [7].
- **Reparameterization Trick:** During training, sample $z=\mu+\sigma\odot\epsilon$ (ε from Normal) to allow gradients.
- **Generation:** After training, sample $z\sim N(0,1)$ and run the decoder to create new images (as shown above). Explore latent arithmetic (e.g. interpolate between encoded images).
- **Resources:** Jackson-Kang's PyTorch VAE tutorial (with code and MNIST examples) is free [8]. Jaan Shervin's VAE blog provides in-depth explanation [6]. PyTorch's official examples and various tutorials (e.g. PyImageSearch, LearnOpenCV) show VAEs in code. Practice by training a VAE on MNIST or CIFAR-10.

3.2 Generative Adversarial Networks (GANs)

GANs consist of two networks trained adversarially: a **generator** $G(z)$ takes noise z and produces fake images, and a **discriminator** $D(x)$ tries to distinguish real images from fakes [9]. Training alternates: D learns to classify real vs. generated images, and G updates its weights (via D's feedback) to produce more realistic images [9] [10].

- **Adversarial Objective:** Originally a minimax game (Goodfellow et al. 2014) where G minimizes and D maximizes a certain loss. In practice, non-saturating variants or Wasserstein losses (WGAN) help stability.
- **Architectural Variants:**
- **DCGAN:** Uses deep convolutional nets for G and D (Radford et al. 2016).
- **Conditional GANs (cGAN):** Provide class labels or other conditioning to generate specific categories.
- **StyleGAN family:** StyleGAN1/2/3 (Karras et al.) introduce style modulation and progressive growing for high-quality faces.
- **Training Tricks:** GANs can suffer from mode collapse and training instability. Techniques like gradient penalty (WGAN-GP), spectral normalization, two-time-scale updates, and careful hyperparameters are important (see literature on GAN training challenges).

- **Implementation:** Both TensorFlow and PyTorch offer free DCGAN tutorials (e.g. TensorFlow's DCGAN tutorial). PyTorch's tutorials and `torchvision` examples include GANs. To practice, build a simple GAN on MNIST or CIFAR-10 and monitor losses/fidelities.
- **Evaluation:** Common metrics include Inception Score (IS) and Fréchet Inception Distance (FID) to quantify image quality/diversity. Use open implementations (e.g. `torch-fid`).

3.3 Diffusion Models

Diffusion models generate images by **reversing** a gradual noising process [11]. In the *forward* diffusion process, Gaussian noise is added to images over many small timesteps until they become indistinguishable from random noise (see illustration above of Gaussian variance increasing). The *reverse* process trains a neural network (often a U-Net) to predict and remove the noise step by step. Ho et al. (2020) showed in DDPMs that this can match GANs in image quality [12]. Modern improvements (DDIM for faster sampling, classifier-free guidance for conditional generation) and **latent diffusion** (applying noise in a learned latent space) have made these models practical for high-resolution and text-conditioned generation.

- **Forward/Reverse Processes:** At training, repeatedly apply $x_{t} = \sqrt{1-\beta_t}\,x_{t-1} + \sqrt{\beta_t}\,\epsilon$ to add Gaussian noise. The model is trained to predict ϵ given x_t. At sampling, start from pure noise and run the learned reverse steps to recover an image [11].
- **Loss:** Reduces to a simple MSE between the true added noise and the model's prediction. Key reference: DDPM (Ho et al. 2020) [12].
- **Latent Diffusion:** Models like Stable Diffusion (Rombach et al. 2022) apply diffusion in a lower-dimensional latent space (via a pretrained autoencoder) [13], drastically reducing compute.
- **Conditional Diffusion:** To generate images from text (or other modalities), incorporate condition guidance. For example, Stable Diffusion uses a frozen CLIP text encoder to steer the diffusion [13]. Google's Imagen uses classifier-free guidance with text encoders.
- **Resources & Implementation:** Hugging Face's Diffusers library (free, open-source) provides examples of training and running diffusion models. Their "Train a diffusion model" tutorial shows how to set up the training loop (see code steps) [14]. The LearnOpenCV tutorial above gives a detailed theoretical overview [11]. Practical exercise: train a simple diffusion model on MNIST, or fine-tune a pretrained latent diffusion (Stable Diffusion) using Hugging Face notebooks.

4. Multimodal Systems

Multimodal AI combines vision, language (and sometimes other modalities) in one model. For example, **CLIP** (OpenAI) uses a dual-encoder: an image encoder and a text encoder trained jointly on ~400M image–caption pairs [15]. Training maximizes the cosine similarity of matching image–text pairs and minimizes it for non-matches. After training, CLIP maps images and text into a shared embedding space. This enables powerful zero-shot capabilities: to classify an image, one computes its embedding and compares it to embeddings of textual class descriptions (e.g. "a photo of a cat" vs "a photo of a dog") [16] [15].

- **Vision-Language Contrastive Models:** CLIP and Google's ALIGN (similar idea) are **dual encoders** trained with contrastive loss [15]. They excel at retrieval, classification, and understanding semantic content across modalities. Hugging Face's Transformers library includes pretrained CLIP models (see CLIPModel pipeline) [15].
- **Text-to-Image Generation:** Systems like **DALL·E**, **DALL·E 2**, **Stable Diffusion**, and **Midjourney** take text prompts and generate images. OpenAI's DALL·E 2 uses a diffusion process guided by CLIP.

Stable Diffusion (CompVis, 2022) is an open latent diffusion model: it encodes text with a frozen CLIP ViT-L/14 and then iteratively denoises a latent image to produce photorealistic outputs [13]. The Stable Diffusion code and model weights are freely available (see CompVis GitHub and Hugging Face). Learning materials: Hugging Face has tutorials on using Stable Diffusion via their Diffusers library; the original Stable Diffusion paper and model card [13] explain the architecture. Try generating images with a free Colab notebook using Stable Diffusion or DALL·E Mini (Craiyon).

- **Vision-Language Tasks:** Many models can **caption images** or answer questions about images. For example, BLIP and Flamingo (CMU/DeepMind) use image encoders + Transformers to perform captioning/VQA. Hugging Face provides pretrained BLIP and Vision-Encoder-Decoder models; you can fine-tune these on datasets like COCO Captions (free dataset) or VQA. Studying encoder–decoder and encoder-only multimodal architectures (e.g. ViLBERT, LXMERT) is useful to understand these.
- **State-of-the-Art Multimodal LLMs:** Google's **Gemini** is a next-generation model explicitly trained on multiple modalities. According to Google, Gemini 1.0 was pretrained simultaneously on text, images, audio, and video [17] [18]. This "natively multimodal" approach means a single model handles vision and language seamlessly. While the model itself is not open-source, its existence underscores the importance of handling multiple data types. (For independent learning, focus on the principles: how to combine embeddings or use cross-attention between modalities.)
- **Practical Work:** Use CLIP for image–text retrieval or classification (Hugging Face has example code). Build a simple image captioner by fine-tuning a vision-language Transformer on a caption dataset (e.g. with PyTorch Lightning). Experiment with Stable Diffusion for creative text-to-image generation (many free notebooks exist). Read the CLIP and Stable Diffusion papers (both open-access) to understand these systems' training objectives.

Summary: This roadmap progresses from the essentials (programming/math) through core generative model techniques (LLMs, GANs, VAEs, diffusion) to advanced multimodal systems. Use the cited free resources (tutorials, courses, code repos, papers) at each stage, and complement theory with hands-on projects (e.g. implement a mini-GPT, train a GAN on MNIST, or generate images with Stable Diffusion). The combination of rigorous concepts and practical experimentation will build a strong foundation in generative AI.

Sources: Hugging Face LLM Course [2] [3] [4]; Google ML Dev (GAN overview) [9] [10]; LearnOpenCV DDPM guide [12] [11]; Jaan Shervin VAE tutorial [6] [7]; Stanford CS224N site (course resources) [5] [1]; OpenAI CLIP blog [16]; Hugging Face CLIP docs [15]; Stable Diffusion repo [13]; Google Gemini announcement [17] [18].

---

1 5 Stanford CS 224N | Natural Language Processing with Deep Learning
https://web.stanford.edu/class/cs224n/

2 3 4 The Large Language Model Course
https://huggingface.co/blog/mlabonne/llm-course

6 7 Tutorial - What is a variational autoencoder? – Jaan Lı 李
https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

8 GitHub - Jackson-Kang/Pytorch-VAE-tutorial: A simple tutorial of Variational AutoEncoders with Pytorch
https://github.com/Jackson-Kang/Pytorch-VAE-tutorial

9 10 Overview of GAN Structure | Machine Learning | Google for Developers
https://developers.google.com/machine-learning/gan/gan_structure

11 12 InDepth Guide to Denoising Diffusion Probabilistic Models DDPM
https://learnopencv.com/denoising-diffusion-probabilistic-models/

13 GitHub - CompVis/stable-diffusion: A latent text-to-image diffusion model
https://github.com/CompVis/stable-diffusion

14 Train a diffusion model
https://huggingface.co/docs/diffusers/en/tutorials/basic_training

15 CLIP
https://huggingface.co/docs/transformers/model_doc/clip

16 CLIP: Connecting text and images | OpenAI
https://openai.com/index/clip/

17 18 Introducing Gemini: Google’s most capable AI model yet
https://blog.google/technology/ai/google-gemini-ai/