Project Report: Asymptote Code Generation with Gemma 3

Project Goal:
The primary objective of this project was to explore the fine-tuning of Google's Gemma 3 large language models for the specialized task of generating Asymptote vector graphics code. The ultimate aim was to enable the model to produce Asymptote code that corresponds to visual diagrams, potentially paving the way for an image-to-code generation system for technical drawings.

1. Dataset Creation: Asymptote Code and Image Pairs
A significant part of this project involved creating a custom dataset tailored for Asymptote code generation. This was achieved using the makeDataset.py script, which automated the following process:

- Source Material: The script processes .asy (Asymptote code) files located in the asymptote-exemples/ directory. An auxiliary script, testtt.py, was also used to crawl https://blog.piprime.fr/asymptote/ (outputting to asyncrawl/) as part of an initial exploration for more Asymptote examples, though makeDataset.py focuses on the local asymptote-exemples/ directory.

- Filtering: Each .asy file's content is checked. Files containing import three (indicating 3D Asymptote modules, which can have more complex rendering requirements) are currently skipped to focus on 2D examples.

- Compilation & Image Generation: For valid 2D .asy files, the script invokes the Asymptote command-line tool (asy) to compile the code and render it as a PNG image. These images are saved into the asy_images/ directory with unique filenames. The script uses a texpath specified as /Library/TeX/texbin (common for macOS TeX Live installations, also configurable via config.asy), which might require adjustment on other systems.

- Code-Image Pairing: The original Asymptote code content is paired with the relative path to its corresponding generated PNG image.

- Dataset Format: The resulting dataset is saved as asymp_dataset.json. Each entry in this JSON file consists of:
    - instruction: A fixed prompt, "Generate the Asymptote code for this diagram."
    - image_path: The path to the rendered PNG image.
    - code: The original Asymptote source code.

- Verification: The verify_match.py script was developed to help validate the dataset. It allows a user to select an entry from asymp_dataset.json, re-renders the Asymptote code from that entry into a new image (in verify_temp/), and facilitates a comparison between the newly rendered image and the one stored in the dataset.

This process successfully built a dataset of (instruction, image, code) triples, forming the foundation for training a model capable of understanding the relationship between visual diagrams and the Asymptote code that produces them.

2. Model Selection and Fine-Tuning Approach

- Model: Google's Gemma 3 series models were chosen, specifically accessed via Unsloth's optimized implementations (e.g., unsloth/gemma-3-27b-it in gemma3Training.py and unsloth/gemma-3-4b-it in testGemma.py). Unsloth facilitates efficient fine-tuning using techniques like LoRA (Low-Rank Adaptation).

- Training Script (gemma3Training.py): This script, derived from Unsloth's example notebooks (originally titled "Copy of Gemma3_(4B)Medical.ipynb"), was intended as the primary script for fine-tuning. It includes Unsloth's FastModel for loading models in 4-bit precision, setting up LoRA adapters, and configuring chat templates (gemma-3).

- Current Training Status: As provided, the gemma3Training.py script has key sections related to data loading (e.g., custom dataset, mlabonne/FineTome-100k, lavita/ChatDoctor-HealthCareMagic-100k), data preparation, and the SFTTrainer training loop commented out. Therefore, in its current state, this script primarily functions to load a base Gemma 3 model and perform inference, rather than fine-tuning on the custom asymp_dataset.json or other datasets mentioned. If fine-tuning was performed, it was likely using a modified version of this script or with the commented-out sections activated (which point towards text-only datasets).

3. Challenges and Outcomes

- Dataset Success: The makeDataset.py script successfully created a bespoke dataset of Asymptote code paired with rendered images, which is a valuable asset for this specific task.

- Multimodal Training Gap: A primary challenge encountered was bridging the custom image-to-code dataset (asymp_dataset.json) with a training pipeline capable of true multimodal fine-tuning. While Gemma 3's architecture (especially as packaged by Unsloth) might inherently include components for multimodal processing (leading to logs about 'vision_tower' during model loading), the provided gemma3Training.py script, in its current partially-commented state, is configured for text-based fine-tuning or base model inference. Actively training the model to understand the visual content of images from asymp_dataset.json and correlate it with Asymptote code requires a more specialized multimodal training setup.

- Inference Capability: The scripts (gemma3Training.py, testGemma.py, newPython.py) successfully demonstrate loading different sizes of Gemma 3 models and performing text-based inference using Unsloth. The model can respond to general prompts and even attempt to generate Asymptote code based on text descriptions (e.g., "Generate Asymptote code for a circle...").

4. Future Work

- Enable Multimodal Fine-tuning: Modify gemma3Training.py (or create a new script) to:
    - Load the asymp_dataset.json.
    - Properly process image inputs (load images, convert to pixel values) and combine them with text prompts.
    - Utilize a training framework (like Unsloth's upcoming multimodal support or custom PyTorch loops) that can effectively train the vision and language components of Gemma 3 on the image-code pairs.

- Experiment with Prompting: For an image-to-code task, the prompt structure would need to clearly signal to the model that an image is provided and Asymptote code is expected.

- Evaluate Performance: Once multimodal fine-tuning is implemented, rigorously evaluate the model's ability to generate accurate Asymptote code from previously unseen diagrams.

- Expand Dataset: Further enlarge and diversify the asymp_dataset.json with more complex Asymptote examples.

Conclusion:
This project has successfully laid the groundwork by creating a specialized Asymptote image-code dataset. While the current training scripts are geared towards text-based interaction, the potential exists to adapt them for true multimodal fine-tuning, which would be the next logical step to achieve the goal of an AI-powered Asymptote code generator that understands visual inputs.