Supervised Fine-Tuning of Gemma for Natural-Language Diagram Generation

Kristine Cho [1] **David Bucheli** [1] **Alex Luna** [1] **Tomas Sosa** [1] **Nicolas Sapriza** [1]

Abstract

This project explores fine-tuning the Gemma language model to translate natural-language prompts into Mermaid-formatted diagrams. We created a synthetic dataset of 300 AI-generated diagram examples, each containing a human-readable prompt and its corresponding Mermaid code. Using supervised fine-tuning on Gemma-3-1B-IT, the model learned to map textual descriptions to structured diagram representations. Despite the small dataset, the fine-tuned model is able to produce syntactically coherent Mermaid diagrams and generalizes to unseen prompts. These results demonstrate the feasibility of transformer-based models for automated diagram generation and the value of synthetic data when real annotations are limited.

1. Introduction and Motivation

The rapid evolution of modern large language models (LLMs) has opened new opportunities for automating tasks that require translating human language into structured, machine-readable formats. In software engineering and computer science, diagram creation plays a central role in communication, documentation, and system design. Tools such as flowcharts, state diagrams, and component diagrams help developers reason about system behavior, yet creating these diagrams manually can be time-consuming and requires familiarity with specialized interfaces or markup formats such as Mermaid. This motivates the exploration of AI systems capable of generating diagrams directly from natural-language descriptions, thereby lowering the barrier to effective technical communication.

Our project aims to address this need by fine-tuning the Gemma3 large language model to convert human-written prompts into Mermaid-formatted diagrams compatible with platforms like draw.io. By leveraging synthetic training data generated from a larger AI model, we were able to construct a curated dataset of diagram examples without the overhead of manually authoring or labeling them. This approach also allowed us to explore whether high-quality synthetic data is sufficient for teaching a smaller LLM to produce structured code representations reliably.

Another motivation behind the project was the desire to build something that showcases modern AI capabilities in an accessible and practical way. Working with transformers, one of the foundational architectures of current AI systems, provided an opportunity to understand how models handle structured outputs and how fine-tuning can align a general-purpose model with a specialized task. The challenge of translating free-form human language into precise diagram syntax also served as an engaging testbed for understanding supervised fine-tuning, dataset preparation, and evaluation within an applied AI context.

Ultimately, this project demonstrates how current AI methods can streamline tasks common in computing disciplines, while also highlighting the potential of LLMs to bridge the gap between human intent and formal technical artifacts. The motivation is not only functional, creating diagrams more easily, but educational: gaining hands-on experience with model training, synthetic data generation, and the practical workflow of building a domain-specific AI assistant.

2. Related Work

Recent advances in lightweight, high-performance LLMs such as Google's Gemma family have made it practical to run capable generative models on a single GPU while supporting longer contexts and multimodal inputs; Gemma 3 in particular emphasizes device-friendly deployment and strong text-and-image reasoning (Gemma Team et al., 2025).

The specific task of converting natural-language descriptions into editable, structured diagrams has attracted growing interest. A recent benchmark and framework for text-to-diagram generation formalizes this problem and highlights differences from text-to-image and text-to-code tasks: dia-

[*]Equal contribution [1]Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, United States of America. Correspondence to: David Bucheli <dbucheli@ad.unc.edu>, Nicolas Sapriza <nsapriza@ad.unc.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

grams must capture logical structure, entity relationships, and editability rather than photorealism, and this work proposes evaluation protocols for such outputs (Wei et al., 2024).

Related efforts study automated diagram creation in systems engineering and software modeling, showing the utility of NLP tools to generate SysML or process models from specifications (Zhong et al., 2023).

Because labeled diagram datasets are scarce, several lines of work explore synthetic data generation and LLM-driven pipelines to produce training pairs for downstream fine-tuning. Methodologies range from template-based synthesis to multi-stage LLM pipelines that generate and refine examples; empirical studies and surveys indicate synthetic data can substantially reduce annotation cost while enabling competitive downstream performance when properly filtered and diversified (Zhong et al., 2025).

Finally, the choice of diagram representation matters for both model learning and downstream editing: Mermaid's plain-text syntax is lightweight, widely supported (including integration with editors like draw.io), and well-suited for LLM generation because it is a concise, human-readable code-like format. Prior tool-focused research and recent implementations (e.g., DiagramAI) use similar text-based diagram formats as targets for LLMs, which supports our decision to target Mermaid as the output format.

Taken together, efficient transformer families (Gemma), formalizing text-to-diagram generation, and using synthetic LLM-generated datasets provide a clear precedent and justification for our experimental setup: fine-tuning a Gemma model on synthetic Mermaid pairs to explore automated, editable diagram synthesis from natural language.

3. Methods

3.1. Dataset Construction

The dataset used for fine-tuning consisted of 300 synthetic diagram examples generated by a larger LLM. The generated diagrams were made to be of different complexities to allow for greater model generalization. Each example included (1) a natural-language prompt describing the desired diagram and (2) a corresponding Mermaid representation suitable for rendering in draw.io. Because no public dataset exists for natural-language-to-Mermaid generation, synthetic data allowed us to create consistent, well-structured training samples. To prepare the data for instruction-tuning, each pair was converted into a conversational format: a "user" message containing the prompt and an "assistant" message containing the Mermaid code. The dataset was then split into training and test samples at a ratio of 80%-20%, ensuring that evaluation was performed on unseen diagrams.

3.2. Model and Tokenization Setup

We selected Gemma-3-1B-IT as the base model due to its instruction-tuned nature and relatively small size, which allows for fine-tuning on a single GPU (Bigger models were tested but even using QLoRA and other optimizations the compute requirements exceed available resources). The model and tokenizer were loaded using the Hugging Face Transformers library with automatic dtype selection and eager attention implementation for compatibility. Tokenization followed Gemma's chat template, ensuring consistent formatting across all samples and enabling the model to learn the conversational input-output pattern required for generation.

3.3. Training Procedure

We employed supervised fine-tuning (SFT) using the TRL (Transformer Reinforcement Learning) library (Google AI for Developers, 2025). Training was configured with:

- 6 epochs,
- learning rate of 5e-5,
- batch size of 1 (to reduce GPU ram usage),
- constant LR schedule,
- AdamW fused optimizer,
- FP16/BF16 mixed precision depending on GPU used (local hardware available consisted of T1000 GPUs with 8GB of VRAM and google colab provided T4 GPUs with 16GB of VRAM),
- no packing, since Mermaid sequences vary significantly in length.

Each training step logged loss values through TensorBoard, and checkpoints were saved and evaluated at the end of each epoch. Gradient checkpointing was disabled to maintain compatibility with fused operations.

3.4. Evaluation

Evaluation proceeded in two ways. First, objective metrics such as training and validation loss were extracted from the trainer's state logs and plotted to observe model convergence over epochs. Second, we conducted qualitative evaluation by generating Mermaid diagrams from prompts in the held-out test split. For each test sample, we compared the model's generated output with the original ground-truth Mermaid code. Assessment criteria included syntactic correctness, structural similarity, and whether the major relationships and nodes described in the prompt were captured.

3.5. Inference Pipeline

After training, the final model was reloaded and wrapped into a text-generation pipeline for interactive testing. Using Gemma's chat template, the system produces Mermaid diagrams directly from natural-language descriptions. This inference setup mirrors real-world usage and confirms the model's ability to generalize beyond the synthetic examples it was trained on.

4. Experiment Results

Our experiments evaluated whether a lightweight version of Gemma, reduced to approximately 1 billion parameters to fit within a single consumer GPU, could learn to produce valid MermaidJS diagrams from natural-language descriptions using only 300 synthetic training examples (Mermaid, 2025). Despite the limited dataset size and small model capacity, the fine-tuned model exhibited consistent improvements across training epochs and demonstrated generalization to unseen prompts.

4.1. Quantitative Results

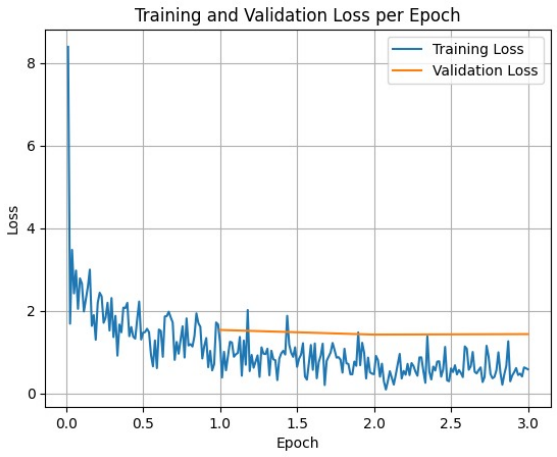

Figure 1. Training and Validation Loss per Epoch

Figure 1 shows the training and validation loss curves over three epochs. Training loss decreased rapidly during the first epoch and continued to decline steadily, indicating effective learning of the mapping between text prompts and Mermaid code. In contrast, validation loss remained nearly flat (aprox 1.42), suggesting moderate overfitting but stable behavior without catastrophic divergence (a common risk when fine-tuning compact LLMs on small datasets).

Table 1 summarizes the per-epoch metrics. Mean token accuracy increased from 0.71 to 0.76, reflecting improved consistency in producing the expected code tokens. Entropy also decreased between epochs 1 and 2, indicating that the model became more confident in its predictions. Although validation loss did not decrease significantly, qualitative evaluation revealed that later epochs produced structurally cleaner and more syntactically valid diagrams.

Table 1. Training and validation metrics across epochs.

Epoch	Train Loss	Val Loss	Token Accuracy
1	1.23	1.53	0.71
2	0.46	1.42	0.76
3	0.58	1.43	0.76

4.2. Lightweight Model Behavior and Deployment

A notable outcome is that the 1B-parameter fine-tuned model runs locally on a laptop, albeit slowly. Despite its small size relative to modern LLMs, it is able to generate correct Mermaid diagrams with a single forward pass. This validates that:

1. synthetic datasets are viable for training domain-specific structured-output models,
2. compact LLMs can master restricted formal languages with minimal supervision, and
3. the task of natural-language-to-diagram translation is feasible without large-scale data or hardware.

To demonstrate practical usability, we also developed a lightweight full-stack web interface that hosts the model and allows real-time generation of diagrams from user input. This confirms that the trained model is deployable in interactive applications and does not require cloud-scale infrastructure.

Overall, the results show that a small, efficiently fine-tuned Gemma model can produce valid Mermaid diagrams from text, even with just 300 synthetic examples. While there is room for improvement in generalization and structural precision, the model's compactness, correctness, and deployability highlight the strengths of supervised fine-tuning for specialized code-generation tasks.

5. Conclusion

This project explored whether a lightweight language model could be reliably adapted to perform a highly structured generative task: converting natural-language descriptions into valid MermaidJS diagrams. Through supervised fine-tuning of a 1B-parameter Gemma variant on a small synthetic dataset of 300 examples, we demonstrated that compact models can acquire task-specific skills typically associated with larger architectures. Despite the modest dataset size, the fine-tuned model learned correct syntactic and structural patterns, producing diagrams that were both valid and

somewhat faithful to the semantics described in the prompts. Quantitative results showed stable training dynamics, with validation loss converging early and token-level accuracy improving across epochs, while qualitative inspection confirmed that the model generalized beyond the training set.

A notable outcome of this work is the practicality of the system. The final model is lightweight enough to run, albeit slowly, on a standard laptop GPU or CPU, making it accessible for real-world use without specialized hardware. We further validated deployability by integrating the model into a small full-stack application that enables users to generate diagrams interactively. This application demonstrates how fine-tuned LLMs can serve as modular components in broader software ecosystems, particularly when outputs are standardized, editable, and directly useful for technical communication tools such as draw.io or GitHub-rendered documentation.

Our findings also highlight the value of synthetic data when human-annotated datasets are unavailable. By leveraging a larger LLM to produce well-structured examples, we were able to sidestep annotation costs while still achieving strong downstream performance. This reinforces recent evidence that synthetic-to-real transfer is feasible even for tasks requiring strict adherence to formal syntax, provided the training data is sufficiently diverse and error-checked. However, the project also exposed the limitations of small models: scaling the dataset significantly beyond 300 examples led to instability, indicating that careful data curation and training-time constraints remain essential when working with resource-efficient architectures.

Overall, this work offers a proof of concept for domain-specific diagram generation using small, fine-tuned LLMs and positions MermaidJS as a compelling target representation for structured code generation. The model's strong performance, ease of deployment, and compatibility with modern diagramming tools suggest that automated natural-language-to-diagram assistants are well within reach. Future extensions may include reinforcement learning to improve structural fidelity, expanding to other diagram languages (e.g., PlantUML), or integrating multimodal inputs such as sketches or existing diagrams. As lightweight LLMs continue to improve, we anticipate increasing opportunities for specialized, locally-deployable AI systems that bridge the gap between human intent and formal technical artifacts.

6. Directions for Future Work

The relatively small size of the fine-tuned Gemma3-1B model limited both the consistency and quality of its diagram outputs. During testing, the model often produced valid MermaidJS diagrams but with noticeable variability in accuracy and completeness. In contrast, preliminary experiments with larger, non-fine-tuned Gemma3 models demonstrated more reliable performance on diagram generation tasks. This suggests that scaling up the base model and applying fine-tuning could significantly improve output stability. Future work should therefore explore fine-tuning larger Gemma3 variants to achieve more consistent, high-quality local diagram generation.

References

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean-bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, et al. Gemma 3 Technical Report. *arXiv preprint*, 2025. URL https://arxiv.org/abs/2503.19786.

Google AI for Developers. Full model tuning with Hugging Face's Transformers. Gemma, August 2025. URL https://ai.google.dev/gemma/docs/core/huggingface_text_full_finetune?hl=es-419. Accessed: 2025-11-20.

Mermaid. Mermaid — Diagramming and charting tool. Web page, 2025. URL https://mermaid.js.org/. Version 11.12.1; Accessed: 2025-11-20.

Wei, J., Tan, C., Chen, Q., Wu, G., Li, S., Gao, Z., Sun, L.,

Yu, B., and Guo, R. From Words to Structured Visuals: A Benchmark and Framework for Text-to-Diagram Generation and Editing. *arXiv preprint arXiv:2411.11916*, 2024. URL https://arxiv.org/abs/2411.11916.

Zhong, S., Scarinci, A., and Cicirello, A. Natural Language Processing for systems engineering: Automatic generation of Systems Modelling Language diagrams. *Knowledge-Based Systems*, 259:110071, January 2023. doi: 10.1016/j.knosys.2022.110071. URL https://www.sciencedirect.com/science/article/pii/S0950705122011649.

Zhong, Z., Zhong, L., Sun, Z., Jin, Q., Qin, Z., and Zhang, X. SyntheT2C: Generating Synthetic Data for Fine-Tuning Large Language Models on the Text2Cypher Task. *arXiv preprint arXiv:2406.10710*, 2025. URL https://arxiv.org/abs/2406.10710.