

COM-304 FM Project Guidelines

<https://github.com/EPFL-VILAB/com-304-FM-project-2026>

EPFL, Lausanne, Switzerland
Spring 2026

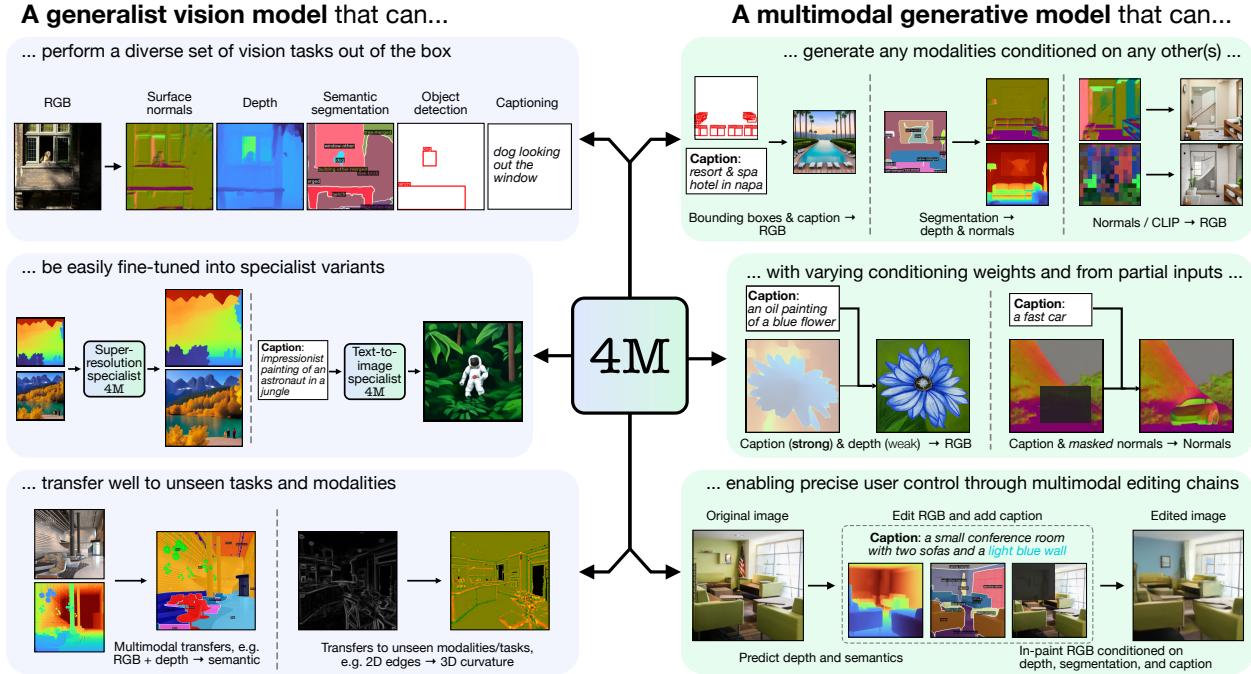


Figure 1: nano4M: In your project you will implement and train a minimal version of a massively multimodal masked model, inspired by 4M [13, 1] and nanoGPT [5]. You will learn what makes current large language and multimodal models work from a practical standpoint.

1 Introduction

The general goal of the project is to understand the building blocks that make up modern (multimodal) large language models (LLMs) like GPT-4o, and assemble them to build vision and language foundation models at a nano scale. This involves building a minimalist multimodal model we call nano4M, inspired by 4M [13, 1], and miniature versions of standard flow matching [9] and VLM models [10, 22]. Although simplified, the principles taught in this project are directly applicable to training real-world large multimodal models.

The project consists of three parts:

- **Building nano4M:** In the first section of the project, you will build a code base for training massively multimodal models, like 4M. First, you will be guided to implement and train an LLM from scratch, learn how to train Transformer-based image generation models, and finally make them multimodal. See section 2 for more information.
- **Advanced Exercises:** Based on the experience you gain from developing a foundation model in the first stage of the project, you will implement 2 additional models: nanoFlowMatching and nanoVLM. See section 3 for more information.

- **Extensions:** Finally, you will develop an extension to nano4M in a group. We will propose a list of extensions of a similar difficulty, and you can either choose to implement one of the listed extensions or propose your own. If you do choose to propose an extension, you must discuss with the TAs to ensure that the extension will be achievable in the four week period. See section 4 for more information.

2 Building nano4M

In this first part of the project, you will build nano4M, a minimal version of 4M[13, 1], inspired by nanoGPT [5]. You will approach this goal in several stages. Detailed instructions will be handed out as the project unfolds.

Building a large language model. Your first task will be to implement an LLM from scratch and train it on a simple dataset to perform text generation. To this end, you will build a causal ("decoder-only") Transformer architecture, and the necessary infrastructure to train it on a text dataset. You will also learn how to perform autoregressive inference. This will be done in the provided nanoGPT exercise.

Building an image generation model. Now that you built and trained an LLM, we will adapt the codebase to support autoregressive [18, 21] and masked image generation [3, 2, 8]. You will learn how to build an "encoder-only" Transformer, about image generation with discrete token-based Transformer models, masked training objectives, and alternative decoding schemes to standard autoregressive generation. This will be included in the nanoMaskGIT exercise.

Building a multimodal model. Finally, we will combine what we have learned into a multimodal model, inspired by 4M [13]. This model will be trained on an aligned multimodal dataset with a multimodal masking objective, which will result in "any-to-any" generation capabilities like the ones shown in fig. 1 (e.g., chained multimodal generation, performing classical vision tasks out-of-the-box, in-painting, etc.). We will provide you pre-trained multimodal tokenizers, as well as the pre-tokenize dataset. You will learn how to implement "Encoder-Decoder" Transformer architectures and train them with a cross-modal masked prediction objective. The codebase implemented at the end of this will be the base for implementing a number of exciting extensions, as discussed in section 4. This will be covered in the nano4M exercise.

3 Advanced Exercises

After building nano4M, you will develop two other common foundation models used today. These will be split into two different exercises that assume knowledge from the nano4M development.

Building a Flow Matching model. In this exercise, the goal is to understand how generative modeling can be formulated as learning a *velocity field* [9, 12] that transports noise to data through an ordinary differential equation (ODE). Instead of generating data token by token, this approach learns how to continuously transform random noise into meaningful samples. You will train a model to guide this transformation and then use a simple numerical solver to generate new data from scratch. Along the way, you will explore practical techniques used in today's large-scale generative models, such as improved timestep sampling and guidance methods for balancing diversity and quality. By the end of the exercise, you will have built a clean and minimal Flow Matching framework and gained intuition about how continuous-time generative models work.

Building a Vision-Language Model (VLM). The nanoGPT that we will build in first part of nano4M project has text tokens in and text tokens out. The resulting model understands and generates text, but what about understanding images alongside text so we can develop visual chatbots? This is the very purpose of this exercise, where we will study how to create a vision-language model that can reason over both

text and images. We will develop nanoVLM and study different aspects including architecture choices, dataset formats, training objectives, and efficient inference techniques. By the end of this exercise, you will understand how to build a vision-language model that can answer questions about any image you provide, essentially providing seeing capabilities to your nanoGPT.

4 Extensions for nano4M

You will choose and/or propose an extension to the basic nano4M to get the full mark. You will have 4 weeks to complete the extension in your group before presenting the final results.

Beyond the proposed set of extensions, we encourage you to come up with novel ideas. The more creative the extension, the better. Creativity can appear in various forms, e.g., in terms of problem selection, solution formulation, implementation, experimentation, or demonstration. We will cast a wide net and will be flexible with any meaningful efforts, so focus your energy on doing something interesting and less on worrying about grades. If you propose your own extension, we will ask you to discuss with the TAs to ensure that the idea is possible in the time you have to complete the extensions.

Below we provide a selection of potential extensions to give you an idea of possible directions you can choose to work on. You can use them directly or as inspiration for your proposal. We give rough estimates for the difficulty and effort required, but not all examples will list them. We will provide more detailed examples closer to the release of the extension proposal template in Week 7.

- *Tokenization:* Current off-the-shelf tokenizers might not yield optimal reconstruction performance for certain modalities. Instead, train specialized tokenizers and compare reconstruction, generation, and alignment performance against the default tokenizer. Also explore the effects of using different losses to train your tokenizer such as reconstruction loss vs FID.

This extension's timeline includes extensively benchmarking currently used tokenizers, and improving tokenizers validated via quantitative evaluations.

- *Super-resolution:* Similar to adding a new modality, super-resolution can be performed as a token-to-token mapping problem, too. In this case, your task is to jointly train on low-resolution images (e.g. 224x224 pixels = 14x14 tokens), and high-resolution images (e.g. 448x448 pixels = 28x28 tokens), teaching the model to map from one to the other.
- *Add a new modality / task:* Extend nano4M to additional data types. This could be any modality you can extract from the existing ones, through known functions (e.g. edge detection), or off-the-shelf neural networks (e.g. feature maps or pseudo labels). Investigate how multimodal alignment, reconstruction, and generation evolve when integrating these diverse modalities into the model. Depending on the modality, this may be paired with implementing and training your own modality-specific tokenizer, or you can experiment with repurposing an existing tokenizer to encode your new modality.

Some modalities are harder to add than others, as they are hard to tokenize, lack data or have other issues. Recommended modalities that would be achievable to add include:

- Semantic segmentation
- SAM [7] instances
- Canny edges
- CLIP [16] or DINO [14] feature maps

This extension's timeline includes pseudo-labeling data for the new modality, training and/or validating the tokenizer, and pretraining your model again with a new set of modalities.

- *Use μP :* Add μP [23] to nano4M and demonstrate if you can transfer hyperparameters swept at a small scale to a larger scale. Demonstrate basin alignment between runs using different learning rates and model widths, and compare to a non- μP baseline.
- *nano4M speed run:* Inspired by nanoGPT speedruns [6], how fast (in terms of tokens seen or wall-clock time) can you train nano4M to a pre-determined validation loss? This may be paired with adding alternative optimizers, like muon [4, 11], or implementing certain architecture modifications. You may also want to look into making your model compatible with `torch.compile`, and adding FlexAttention [15].
- *Alternate masking strategies:* In this extension, you will ablate on different masking strategies for training nano4M. Some of such strategies can include span-masking [17] for text-like modalities, which allows for generation of masked-out regions in sequences in an autoregressive manner. Or block-masking in images where large portions of the image-like modalities are masked all together instead of fully random masking. The goal is to understand how different masking techniques during the training can effect the model's performance from various aspects.

You will be provided with a sequential data modality (such as captions) to test your masking with.

- *Architecture improvements:* Investigate potential directions for enhancing the nano4M from the architecture perspective. Perform architecture improvements, such as replacing the MLP by SwiGLU [19], replacing the learned absolute positional embeddings by Fourier or RoPE [20], studying better weight initialization strategies, etc.

In this extension, you will compare the proposed architectural designs for nano4M both qualitatively and quantitatively. For instance, quantitative evaluations could be done using image generation performance metrics such as FID across different model designs. You can also compare other tasks like segmentation or depth estimation.

- *Test-time exploration/scaling and inference strategies:* Most of the above extensions involve design choices at the finetuning/pretraining stage. In contrast to these, there is an interesting area of research called test-time search/scaling, where one can achieve further improvements with the model at test time. For instance, using look-ahead and beam search coupled with a verifier model to improve generation quality (more details on test-time search can be found here). Also, chained predictions are another viable test-time scaling approach, where a mapping from modality A to modality B is decomposed into A \rightarrow X \rightarrow B, with X referred to as the chaining modality. nano4M naturally supports chaining. Please refer to Fig. 3 in [13] for more details on chained predictions.

From the perspective of inference strategies, this extension would involve looking for the best algorithms for generation. This includes but is not limited to comparing iterative decoding vs. one-pass decoding, experimenting with advanced sampling methods (e.g., top-k, nucleus sampling), exploring controlled generation, and measuring trade-offs in speed, reconstruction quality, and sample diversity.

5 Evaluation

Your project and models will be evaluated along several complementary dimensions, based on the quality of its delivery, implementation, results, and the performance of the proposed extensions.

Quality of the solution

1. **Hypothesis-driven research:** Strong projects articulate clear hypotheses or research questions and then design experiments to test them systematically.
2. **Technical Rigor:** Your method and results should be described in enough detail, using figures and sample outputs where applicable, to allow for proper evaluation. Compare your proposed method to appropriate external and self-baselines and perform controlled ablations to demonstrate the impact of your key design choices. Claims in your report about your method/design choices should be supported by quantitative and/or qualitative evidence.
3. **Results:** Model outputs (e.g., quantitative analyses and qualitative examples) must convincingly demonstrate your claims. If you were unable to achieve the results you worked towards and expected, demonstrate an understanding of why that may be the case and what steps you would take to make it work.

Quality of the delivery

1. **Problem Framing & Motivation:** Motivate concisely what you are solving and why it matters.
2. **Clarity & Structure:** The report should be well organized into abstract, introduction, related work, method, experiments, and conclusion sections. See the report guidelines below for more details. Be precise in your writing and avoid purple prose.
3. **Visuals:** Use figures and plots with informative captions to demonstrate your results and to visually convey how your method works. Figures and captions should be able to stand on their own, without requiring extensive back-and-forth between the main text and the figures.

Code verification

1. **Reproducibility:** Submitted code will be evaluated for reproducibility, ensuring that reported results can be independently verified.
2. **Documentation and Clarity:** Provide complete and clear documentation, including a self-contained README, listing all dependencies, setup instructions, and execution details. Ensure that your code is well-organized and thoroughly commented to facilitate easy verification.

6 Reports Instructions

During the project, you will need to submit two reports and make the final presentation of your project. Below, you can find instructions for each stage.

6.1 Extension Proposal

This proposal will be focused primarily on the extension that will be made to the model. You should assume that you have access to all the models developed in the exercises (LLM, image generation model, nano4M, Flow Matching model and VLM) that will be developed in the first half of the project and the extensions will be built on top of those base models.

You should describe each extension you will implement in detail, both at a high level, considering the following questions:

1. **What** is the problem you want to solve?
2. **Why** is it important that this problem be solved?
3. **How** do you solve this problem?

and at a lower level, considering implementation specifics such as:

- Will you need extra data for the model to generalize well with this extension?
- What model architectures will/won't this extension work with?
- How much extra compute might you need to make an extension work?
- And else anything relevant to your specific extensions.

Your description of the extensions should be self-sufficient, so be as specific as possible. You also need to set a plan of what you aim to achieve each week to ensure gradual and sufficient progress in the limited available time. Make sure to assess what can go wrong and discuss other options you can try. The proposal document should be **at least one, and at most two pages** long. Please use the template that we will provide on Moodle in the corresponding week.

Additional words of advice:

- Try to communicate and motivate your idea using visuals, diagrams, charts, or any other appropriate tools you see fit.
- Allocate your time well between the presentation and the final project delivery date.

6.2 Final Presentation Guidelines

Each group will have **5 minutes** to present their system with another **2 minutes** set aside for questions. You should expect the audience to have some general understanding of the base nano4M model and advanced exercises, approximately equivalent to the content covered in the lectures. Hence, the greatest focus should be on your implemented extension. A general outline for the final presentation is highlighted below:

- Introduction & Motivation: Similar to the project proposal guideline, why is your project useful? What problem is your project addressing?
- Method Overview: Give a high level explanation of your method and model. A method figure can be useful to illustrate to the rest of the class what you have built.
- Technical Section: Highlight one or two key aspects of your design. These can be parts you found interesting or particularly difficult to implement.
- Results: Show the results of the extension you proposed to tackle, as well as any interesting results or takeaways you had from the project (eg. videos, images, plots, numbers, etc).

We expect **each team member** to present an **approximately equal portion** of their presentation.

6.3 Final Report Guidelines

Report: Your final report should be **4 pages maximum**, excluding references. You can use appendices without any limits on the page number, but make sure that the main material is provided in the main report. Please use the template that is provided on Moodle alongside these guidelines when writing your report. As before, one submission per group is sufficient. We also ask you to submit your **code** with proper running instructions (e.g. a ReadMe file) in a **zip file**. The ReadMe file should be self-contained: specify the packages to install, their version numbers, commands on how to run your code to reproduce your results and the file hierarchy with a description of each file. Please make sure the code is understandable, e.g. through documentation.

Please try to organize your report in the following suggested way for a better understanding.

- **Abstract:** Provide a brief description of your problem, approach, and key results.
- **Introduction:** Describe the problem you are solving, its significance (i.e. why are you solving it?), and how do you solve it. You can organize this section similar to the introduction of your proposal report while being *more concrete and specific*.
- **Related Work:** How is this problem currently solved (if solved)? Discuss the relevant works to your project and pose your approach against them. Indicate the *differences* and *similarities* between your project and these works as clearly as possible.
- **Method:** Explain your approach for solving the problem. Justify the design choices you made and mention other alternatives, if any. Make sure to include figures, diagrams, pseudo-code, etc. to strengthen your case. It is important that your method is explained in an understandable way for a fair evaluation.
- **Experiments:** Discuss your experimental setup in detail. Explain your baselines and justify why you picked them. Support your results with *quantitative and qualitative evaluations* comparing your method to these baselines (e.g. include tables for performance metrics and qualitative figures.). If relevant, perform ablation studies to provide further insight into the inner workings of your method.

If your project *did not work as expected*; and you instead managed to systematically invalidate an apriori sensible hypothesis; that is also a perfectly meaningful contribution. If this is the case, provide a detailed and sensible analysis that identifies the main modes of failure of your original hypothesis, discusses their potential reasons, and distills what one can learn from them.

The projects will not be regarded as successful only if they “work”; any project that extracts interesting insights or contributes a signal towards evaluating a meaningful hypothesis will also be regarded as successful. The main evaluation criteria will be the degree of creativity, motivation, and scientific rigor with which you managed your project (e.g., asking interesting and sensible questions, forming and validating hypotheses in a systematic way using the appropriate baselines), rather than the end score you obtained on some pre-defined benchmark.

- **Conclusion and Limitations:** Provide a brief summary of and takeaways from your project. Mention the limitations of your method and how can they be solved. Also mention possible future extensions or other use cases.
- **Individual contributions:** If it is a group project, include an author contribution section explaining the role of each group member throughout the project. You can refer to this [exemplary author contribution statement](#) to give you an idea.

Additional suggestions:

- Make sure you proofread your report (or ask an external person to do it, if possible).
- Visuals (figures, tables, etc.) can be more effective at conveying information than writing. But do make sure that your visuals are helpful, e.g. include captions that describe and explain the take home message for your figures and tables, plots are understandable (e.g., with proper labels and readable font size for axes, etc).
- You can include videos as part of your results if it is relevant to your extensions. You can also provide extended image results in your appendix.

Website: In research nowadays it is common to include multiple formats of presenting your project, such as videos, posters and websites. The presentation of this project in particular benefits greatly from displaying a wider range of results in a more accessible way than possible with a report alone. As such, you will also prepare a website / blog post that can display examples of the results of your model. **This website should contain** a short summary of the key takeaways from the project, an extended presentation of what the model is capable of, and what additional capabilities the developed extensions provide the model. You may also choose to add other material such as a video or a demo embedded in the website if you think that may aid the presentation of your project.

Please ensure your website adheres to the following requirements:

- If you use an online report (as opposed to sending the offline webpage package), please use a platform where meeting the deadline can be verified, e.g. use GitHub Pages by creating a repository where the last commit is no later than the deadline.
- Put a corresponding amount of text as to what the page limit for the standard PDF report would allow, i.e. 5 pages excluding references, by using a comparable number of words.

See some examples [here](#) (from CS-503 Spring 2023), [here](#), [here](#), and [here](#) for inspiration.

Good luck! ♡

References

- [1] Roman Bachmann, Oğuzhan Fatih Kar, David Mizrahi, Ali Garjani, Mingfei Gao, David Griffiths, Jiaming Hu, Afshin Dehghan, and Amir Zamir. 4M-21: An any-to-any vision model for tens of tasks and modalities. *Advances in Neural Information Processing Systems*, 2024.
- [2] Huiwen Chang, Han Zhang, Jarred Barber, AJ Maschinot, José Lezama, Lu Jiang, Ming Yang, Kevin P. Murphy, William T. Freeman, Michael Rubinstein, Yuanzhen Li, and Dilip Krishnan. Muse: Text-to-image generation via masked generative transformers. *ArXiv*, abs/2301.00704, 2023.
- [3] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T. Freeman. MaskGIT: Masked generative image transformer. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11305–11315, 2022.
- [4] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024.
- [5] Andrej Karpathy. nanoGPT: The simplest, fastest repository for training/fine-tuning medium-sized GPTs. <https://github.com/karpathy/nanoGPT>, 2023. Accessed: 2025-02-17.
- [6] Jordan Keller. modded-nanogpt: A modified version of nanoGPT with additional features. <https://github.com/KellerJordan/modded-nanogpt>, 2024. Accessed: 2025-02-17.
- [7] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4015–4026, 2023.
- [8] Tianhong Li, Huiwen Chang, Shlok Kumar Mishra, Han Zhang, Dina Katabi, and Dilip Krishnan. MAGE: Masked generative encoder to unify representation learning and image synthesis. *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2142–2152, 2022.
- [9] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.
- [10] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.
- [11] Jingyuan Liu, Jianling Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Meng Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for llm training. 2025.
- [12] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.
- [13] David Mizrahi, Roman Bachmann, Oğuzhan Fatih Kar, Teresa Yeo, Mingfei Gao, Afshin Dehghan, and Amir Zamir. 4M: Massively multimodal masked modeling. In *Advances in Neural Information Processing Systems*, 2023.
- [14] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

- [15] PyTorch Team: Horace He, Driss Guessous, Yanbo Liang, Joy Dong. FlexAttention: The Flexibility of PyTorch with the Performance of FlashAttention, August 2024.
- [16] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.
- [17] Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2019.
- [18] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. *ArXiv*, abs/2102.12092, 2021.
- [19] Noam M. Shazeer. Glu variants improve transformer. *ArXiv*, abs/2002.05202, 2020.
- [20] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.
- [21] Peize Sun, Yi Jiang, Shoufa Chen, Shilong Zhang, Bingyue Peng, Ping Luo, and Zehuan Yuan. Autoregressive model beats diffusion: Llama for scalable image generation. *arXiv preprint arXiv:2406.06525*, 2024.
- [22] Luis Wiedmann, Aritra Roy Gosthipaty, and Andrés Marafioti. nanovlm. <https://github.com/huggingface/nanoVLM>, 2025.
- [23] Greg Yang, J. Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub W. Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *ArXiv*, abs/2203.03466, 2022.