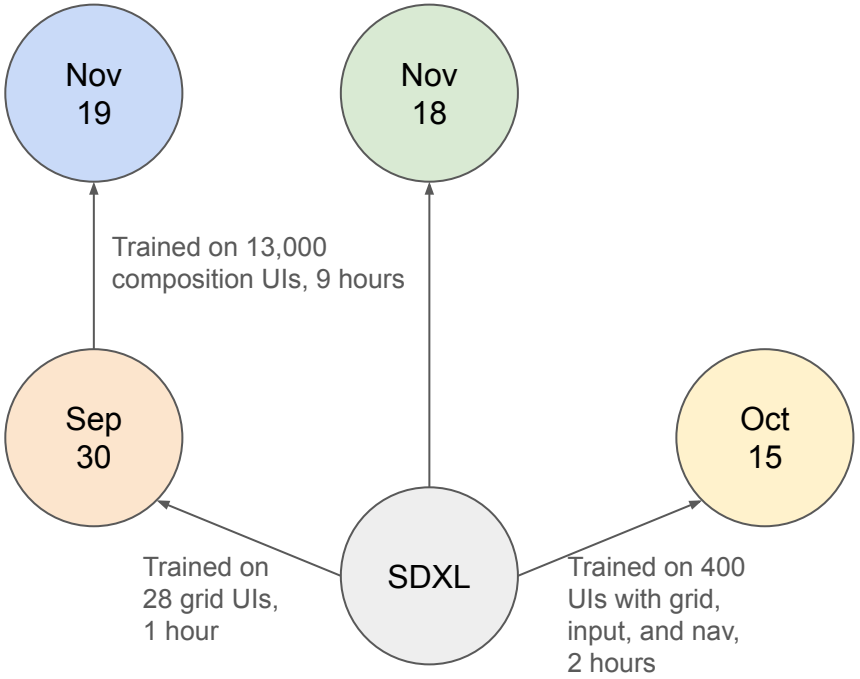
Dec 3rd - Evaluating Fine Tuned Models

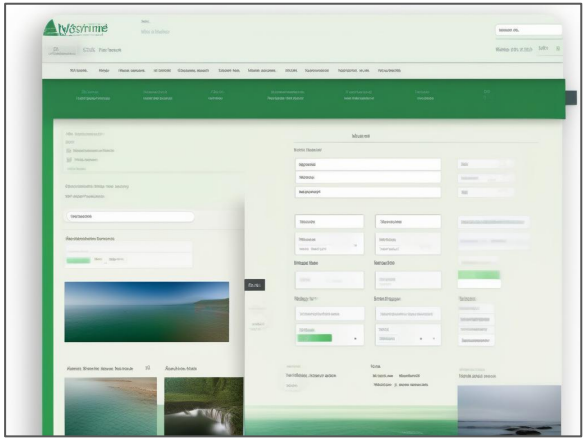
Evaluation

1. Has our fine-tuning work this semester shown improvement over the base model?
2. Which prompts and tasks has fine-tuning improved? Which haven't?
3. What fine-tuning approaches have succeeded and what haven't?

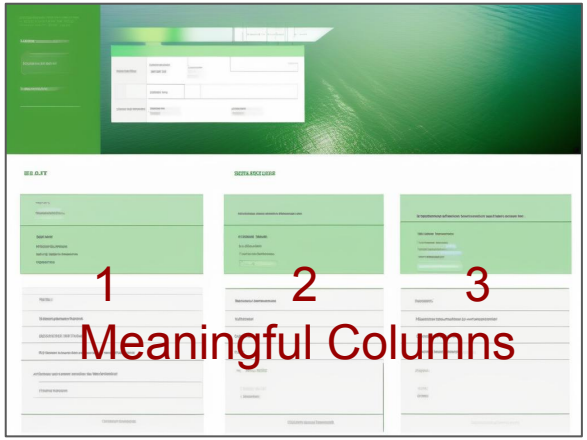


Test 1: Has our fine-tuning work this semester shown improvement over the base model?

Our fine-tuned models are **more likely** to give you reasonable user interfaces, but are still far too **chaotic** to follow specific prompt details.

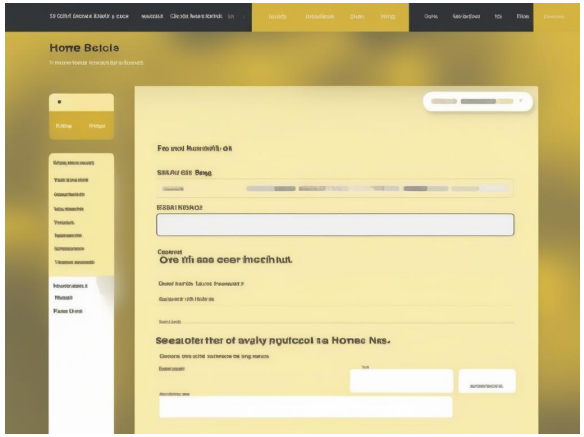


a website user interface with a light theme with green accents with a navigation bar at the top. The navigation bar has a "WATER" logo on the left, three navigation links: ['Home', 'Pricing', 'About'] and a searchbar on the right. The main content of the page is 3 columns each with a date input field, a password input field, and a submit button input. There is an image of water that is the background for the website

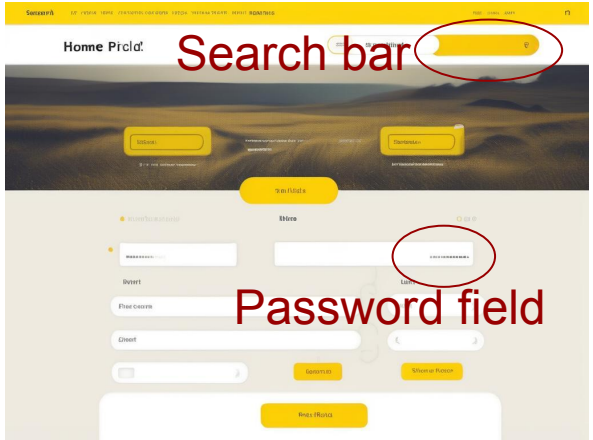


SDXL

Nov 19



a website user interface with a yellow theme with a navbar at the top. The navbar has a logo on the left, three navigation links: ['Home', 'Pricing', 'About'] and a searchbar on the right. The main content of the page is 1 column with a date input field, a password input field, and a submit button input.



Test 2: Which prompts and tasks has fine tuning improved? Which haven't?

After fine tuning, models **rarely blur or overlap** components like the base model. Column and row **layouts are more reasonable** and the model has a slightly **better understanding of different input types and navigation components** specified by the user.

However, putting **input fields together into a form** based on a prompt has not been possible. This is likely because our training images only include input fields one at a time, not composed into forms and specified in the training label. This is an area for future improvement.

Overall layout control is still lacking, as the user cannot force the model to give them their desired number of columns, for example.

Prompting the models with **a negative**, such as "no navbar", has not succeeded.

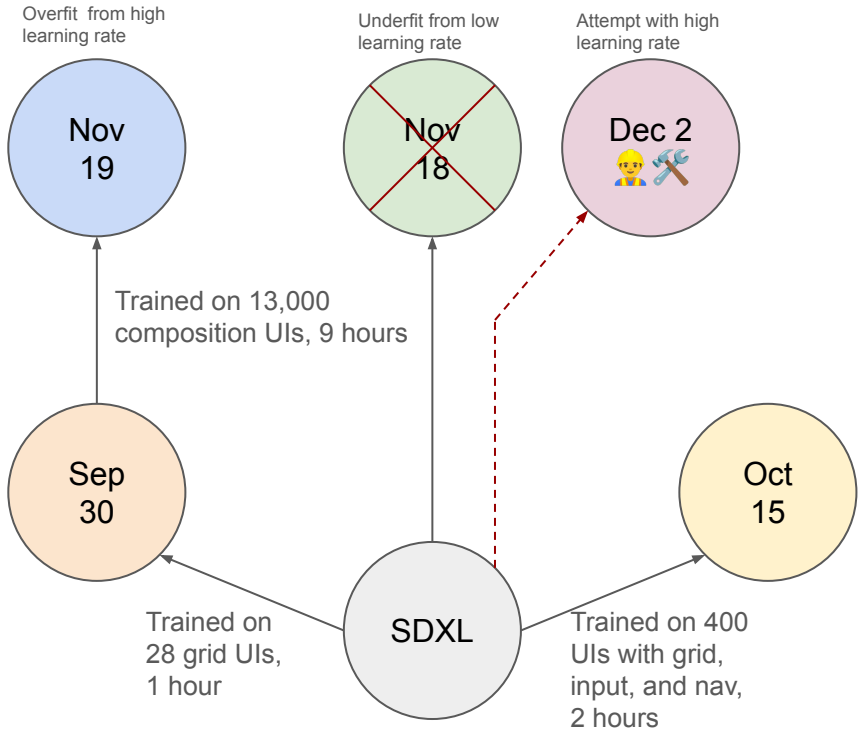
Multiple tries are needed and the user still often receives outlandish results

Test 3: What fine-tuning approaches have succeeded and what haven't?

Isolating concepts when generating training data was very useful. The greatest success we have had was in training the Sep 30th model on a small set of simple grid layouts that could teach a given concept.

It appears that the Nov 19th model showed similar capabilities to the Sep 30th model in generating simple grid UIs. This means that the **first stage progress was not lost or forgotten in stage two of fine-tuning.**

I am currently training a new model (Dec 2) only on the composition UI dataset to **compare** with Nov 19, which was first trained on a simple concept and then trained on the composition dataset. This may illustrate whether the first stage was necessary or whether it can be skipped and only complex UIs are needed.



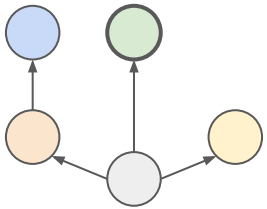
Questions

1. Hugging face allows me to download the dataset as a .arrow or .parquet file, but I can't find a built-in way to download and recover individual screenshot and caption files for use in training our model. Am I missing something?
2. Preparing the latents for model training took about 2 hours, is this reasonable?
3. Fine tuning with our process relies on scripts from this repo of stable diffusion scripts. Should I attempt to extract them for our own repo (with citation) or should I simply mention in the readme that users should also clone that repository?

Next Steps

1. Create my final model and document the steps that were used in training it. Then upload this to Civit.ai.
2. Document which steps in the Civit.ai fine-tuning tutorial are irrelevant to our use-case. This should make it easier for future users to jump in and fine tune.

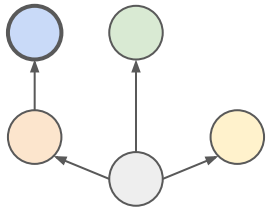
Appendix - Fine Tune Parameters - Nov 18



Accelerate launch
—num_cpu_threads_per_process 1
~/sd-scripts/sdxl_train.py
—pretrained_model_name_or_path=/home/nolan-kyhl/Sandbox/fineTune/**sdXL_v10VAEFix.safetensors**
—in_json ~/Sandbox/fineTune-nov18/meta_lat.json
—train_data_dir=/home/nolan-kyhl/Sandbox/fineTune-nov18/complete/
—output_dir=/home/nolan-kyhl/Downloads/Easy-Diffusion-Linux/easy-diffusion/models/stable-diffusion/
—train_batch_size=4
—learning-rate=1e-6
—max_train_steps=3200
—gradient_checkpointing
—mixed_precision=bf16
—save_every_n_steps=400
—save_model_as=safetensors
—keep_tokens=255
—optimizer_type=adafactor
—optimizer_args scale_parameter=False relative_step=False warmup_init=False
—cache_latents
—lr_warmup_steps=100
—max_grad_norm=0.0
—max_data_loader_n_workers=1
—persistent_data_loader_workers
—full_bf16
—lr_scheduler=constant_with_warmup

Around 8 hours, saving every 400 steps (1 hour).

Appendix - Fine Tune Parameters - Nov 19



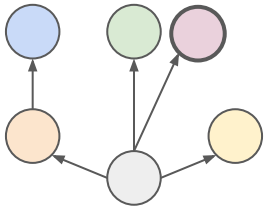
Accelerate launch
—num_cpu_threads_per_process 1
~/sd-scripts/sdxl_train.py
—pretrained_model_name_or_path=home/nolan-kyhl/Downloads/Easy-Diffusion-Linux/easy-diffusion/models/stable-diffusion/**fineTune-sep-30-last.safetensors**
—in_json ~/Sandbox/fineTune-nov18/meta_lat.json
—train_data_dir=/home/nolan-kyhl/Sandbox/fineTune-nov18/complete/
—output_dir=/home/nolan-kyhl/Downloads/Easy-Diffusion-Linux/easy-diffusion/models/stable-diffusion/
—train_batch_size=4
—learning-rate=1e-5
—max_train_steps=3200
—gradient_checkpointing
—mixed_precision=bf16
—save_every_n_steps=400
—save_model_as=safetensors
—keep_tokens=255
—optimizer_type=adafactor
—optimizer_args scale_parameter=False relative_step=False warmup_init=False
—cache_latents
—lr_warmup_steps=100
—max_grad_norm=0.0
—max_data_loader_n_workers=1
—persistent_data_loader_workers
—full_bf16
—lr_scheduler=constant_with_warmup

Initially I thought Nov 18 underfit, so I bumped the learning rate up a factor of 10.

I also began my training not from the base SDXL model, but my Sep 30 model which had been trained on simple grid UIs. This is because I thought it would better handle the more complex UIs if it first had the foundation of simpler concepts baked in already.

This model heavily overfit after that amount of training, however earlier checkpoint models may prove effective.

Appendix - Fine Tune Parameters - Dec 3



Accelerate launch
—num_cpu_threads_per_process 1
~/sd-scripts/sdxl_train.py
—pretrained_model_name_or_path=/home/nolan-kyhl/Sandbox/fineTune/**sdXL_v10VAEFix.safetensors**
—in_json ~/Sandbox/fineTune-nov18/meta_lat.json
—train_data_dir=/home/nolan-kyhl/Sandbox/fineTune-nov18/complete/
—output_dir=/home/nolan-kyhl/Downloads/Easy-Diffusion-Linux/easy-diffusion/models/stable-diffusion/
—train_batch_size=4
—learning-rate=1e-5
—max_train_steps=3200
—gradient_checkpointing
—mixed_precision=bf16
—save_every_n_steps=400
—save_model_as=safetensors
—keep_tokens=255
—optimizer_type=adafactor
—optimizer_args scale_parameter=False relative_step=False warmup_init=False
—cache_latents
—lr_warmup_steps=100
—max_grad_norm=0.0
—max_data_loader_n_workers=1
—persistent_data_loader_workers
—full_bf16
—lr_scheduler=constant_with_warmup

Around 5 hours, saving every 400 steps.

Meeting Notes

It should be possible to obtain raw images and labels from hugging face. You may need to upload as a tar file.

Feel free to extract the script you use from kohya/sd-scripts and cite it so that its contained in your repo.

At this stage, it is important to document and tidy up everything so it is easily reproducible.

It is impressive this project even works given the small amount of input data and old hardware.

2 hours to prepare latents doesn’t seem unreasonable given the dataset size.

Return the hardware whenever, but change the password and give Professor access. Document process.

I can include PDFs of these research updates in the GitHub repo.