intel

OpenVINO™

CVPR 2024

Edge-Optimized Deep Learning: Harnessing Generative AI and Computer Vision with Open-Source Libraries

Module 2:

Customize and Run Gen AI Pipelines with LoRA and OpenVINO™

Fiona Zhao, Zhuo Wu

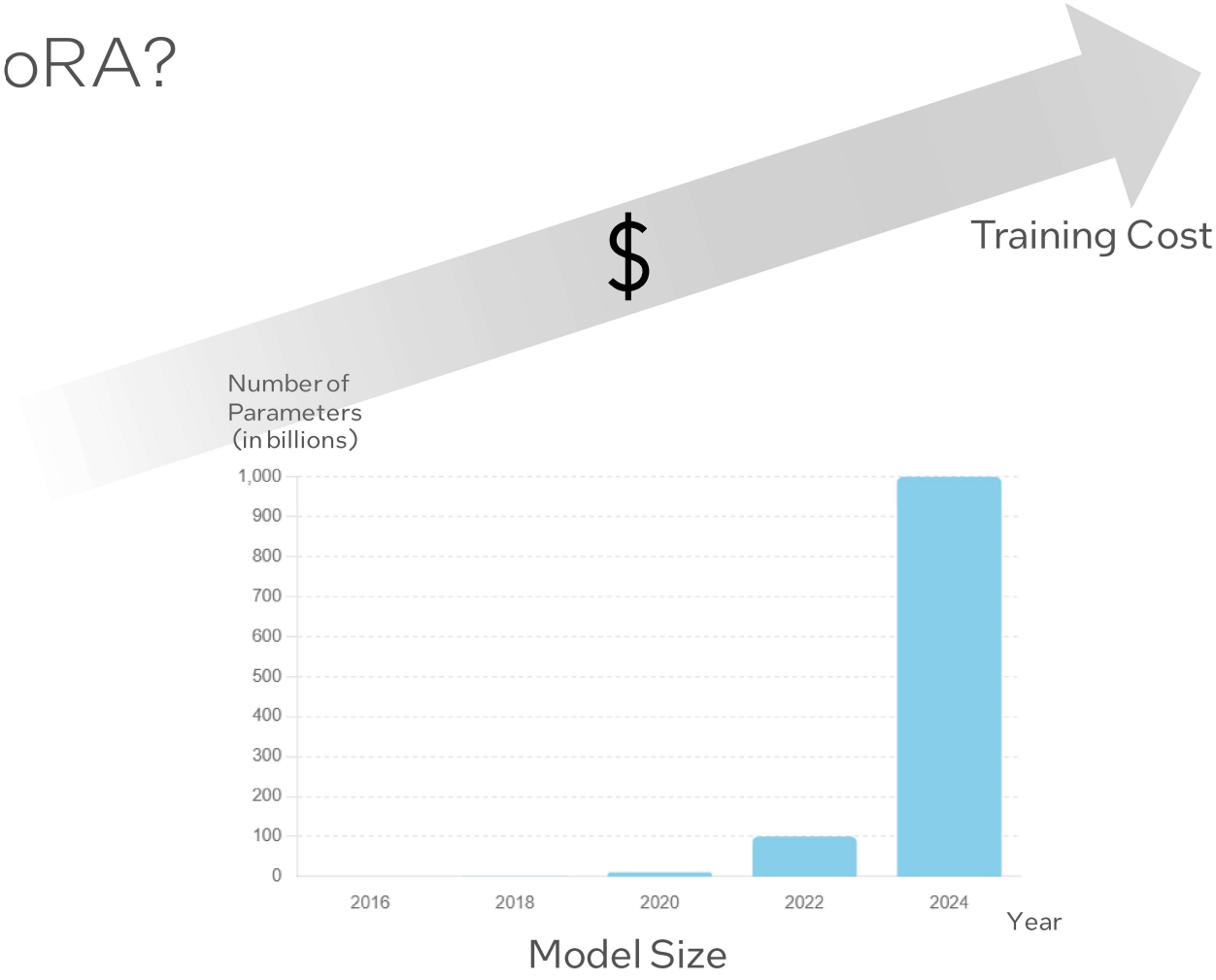
Intel NEX China

Outline

• Show a quick demo of LoRA fine-tuned results	5 minutes
• What is LoRA?	5 minutes
• OpenVINO acceleration on Visual Gen AI pipeline	
◦ Stable Diffusion ControlNet Pipeline	10 minutes
◦ Enable LoRA offline	10 minutes
◦ Enable LoRA weights on OV model in Runtime	30 minutes
▪ Enable LoRA in Runtime	
▪ Enable multiple LoRA weights	
• Hands-on practice	
◦ Pipeline workflow explanation	10 minutes
◦ LCM with LoRA pipeline (C++) packed in .exe to run	20 minutes
• Conclusions	

Placeholder for a demo video or live demo

Why use LoRA?


$
Training Cost
Number of Parameters (in billions)
1,000
900
800
700
600
500
400
300
200
100
0
2016
2018
2020
2022
2024
Year
Model Size

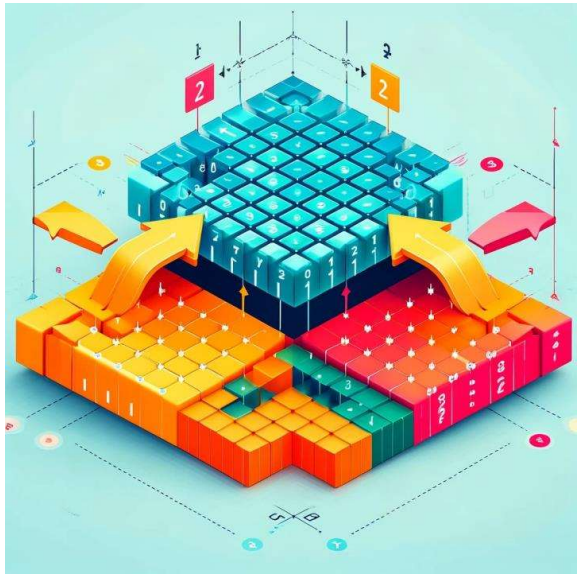
What is LoRA?

Matrix A_{mn}

$$rank(A) \leq \min(m, n)$$

Change of Weights $\Delta W_{n \times k}$

$$rank(\Delta W_{n \times k}) \ll \min(n, k)$$



LoRA for text-to-image models

- LoRA weights can be categorized depending on the concept they are trained
  - Character
  - Style
  - Concept
  - Pose
  - Clothing
  - Object

Find LoRA resources from: https://civitai.com/

Enable LoRA for text-to-image models

Original LoRA enabling from Diffusers:

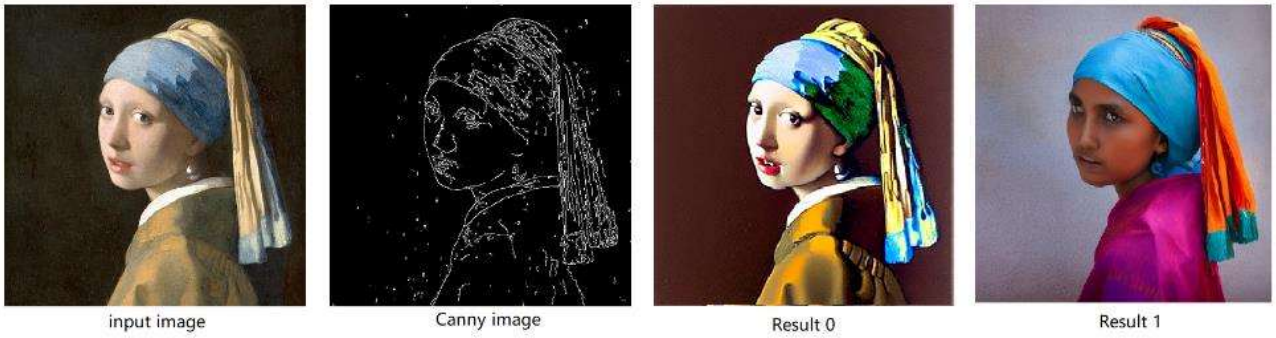
```
pipeline.unet.load_attn_procs(lora_model_path.bin)
```

Supporting A1111 themed LoRA checkpoints from Diffusers:

```
pipeline.load_lora_weights(".", weight_name="light_and_shadow.safetensors")
```

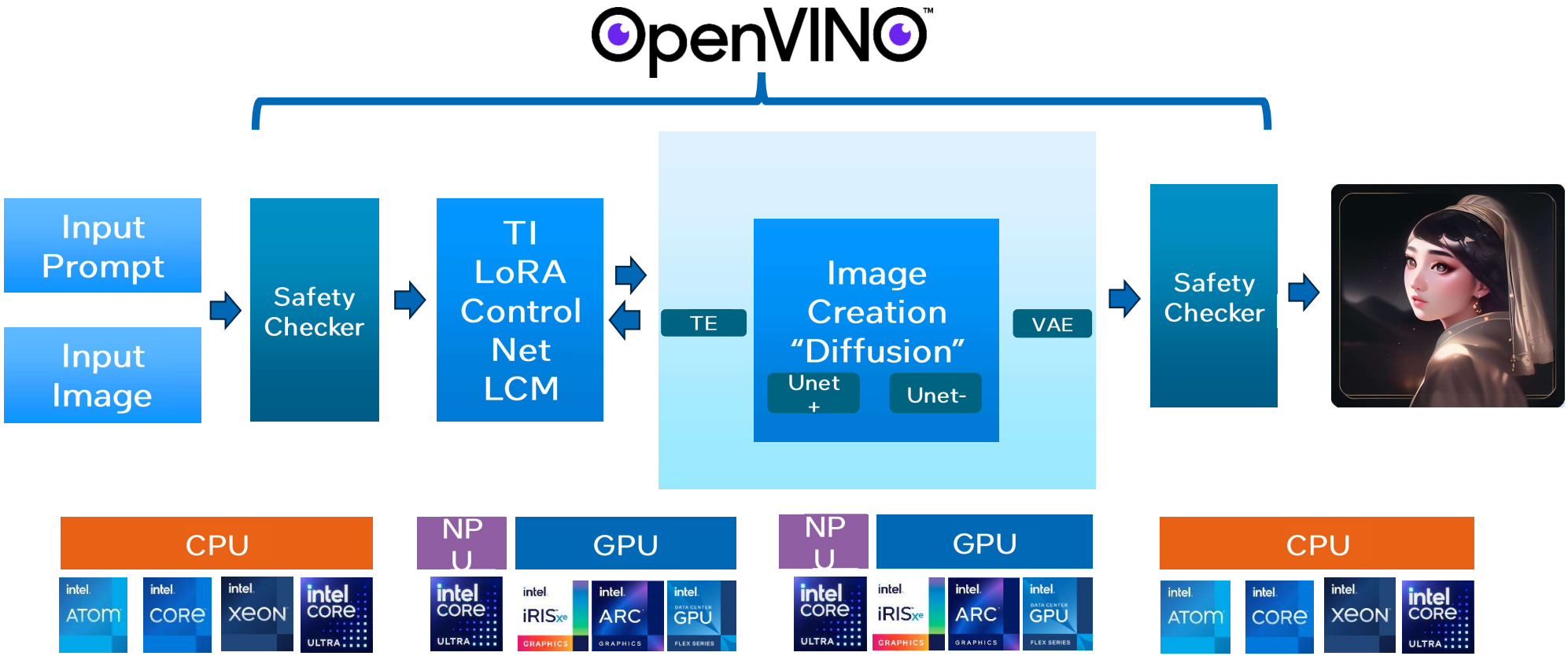
Stable Diffusion ControlNet Pipeline

- Build the Stable Diffusion ControlNet Pipeline
  - Step 1: Environment preparation
  - Step 2: Model Conversion
  - Step 3: Runtime pipeline test



Generated images with original Stable Diffusion v1.5 + canny ControlNet

OpenVINO™ acceleration on text-to-image pipeline



Enable LoRA on OpenVINO™ model Offline

- Load Stable Diffusion ControlNet Pipeline from Diffusers (in PyTorch)
- Load LoRA weights
- Fuse LoRA weights with UNet

Benefits:
easy to use with
Diffusers API

```
# load controlnet model
controlnet = ControlNetModel.from_pretrained(controlnet_id)
# load stable diffusion pipeline
pipe = StableDiffusionControlNetPipeline.from_pretrained(stable_diffusion_id, controlnet=controlnet)
# load LCM LoRA weights
pipe.load_lora_weights(adapter_id)
# fuse LoRA weights with UNet
pipe.fuse_lora()
text_encoder = pipe.text_encoder
text_encoder.eval()
unet = pipe.unet
unet.eval()
vae = pipe.vae
vae.eval()
del pipe
gc.collect()
return controlnet, text_encoder, unet, vae
```

What LoRA weights file (.safetensors) contains:

```
lora_te_text_model_encoder_layers_0_<mlp_fc1|self_atten_k/q/v>.weights
…
lora_unet_<up|down>_blocks_0_attentions_0_transformer_blocks_0_attn1_to_
<k|q|v>.weights
...
```

Reference:
https://github.com/openvinotoolkit/openvino_notebooks/blob/latest/notebooks/latent-consistency-models-image-generation/lcm-lora-controlnet.ipynb

Enable LoRA on OpenVINO™ model Offline

- Convert model to OpenVINO IR

```
ov_model = ov.convert_model(controlnet, example_input=inputs, input=input_info)
```

```
ov_model = ov.convert_model(wrapped_unet, example_input=inputs)
```

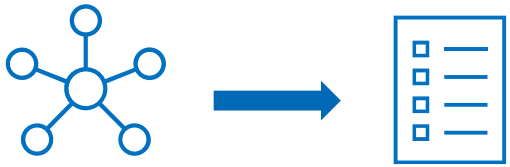
```
ov_model = ov.convert_model(
    text_encoder,  # model instance
    example_input=input_ids,  # inputs for model tracing
    input=([1,77],)
)
```

```
ov_model = ov.convert_model(vae_decoder, example_input=latents, input=[-1, 4, -1, -1])
```

Drawbacks:
N * LoRA theme files → **N** * unet/text_encoder models regeneration → **N** * conversion to OpenVINO IR
Instead of
N * LoRA theme files →**1** * unet/text_encoder models regeneration + **N** * LoRA files for N themes → **1** * conversion to OpenVINO IR

Reference:
https://github.com/openvinotoolkit/openvino_notebooks/blob/latest/notebooks/latent-consistency-models-image-generation/lcm-lora-controlnet.ipynb

Enable LoRA on OpenVINO™ model in runtime



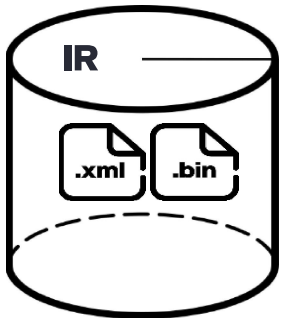
Weights layer name

Decomposition weights

- Workflow:

1. Extract safetensors files, create Dict list to save weights layer name and pairs of decomposition weights.

Enable LoRA on OpenVINO™ model in runtime

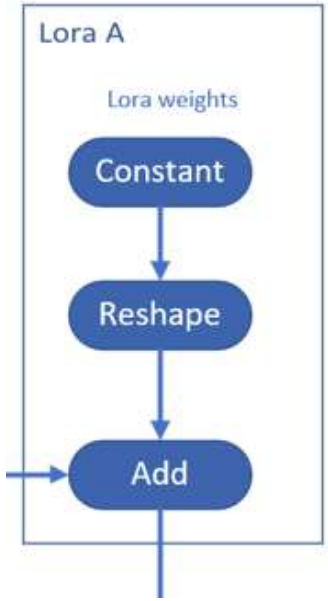


FP16

- Workflow:

2. Read IR models.

Enable LoRA on OpenVINO™ model in runtime



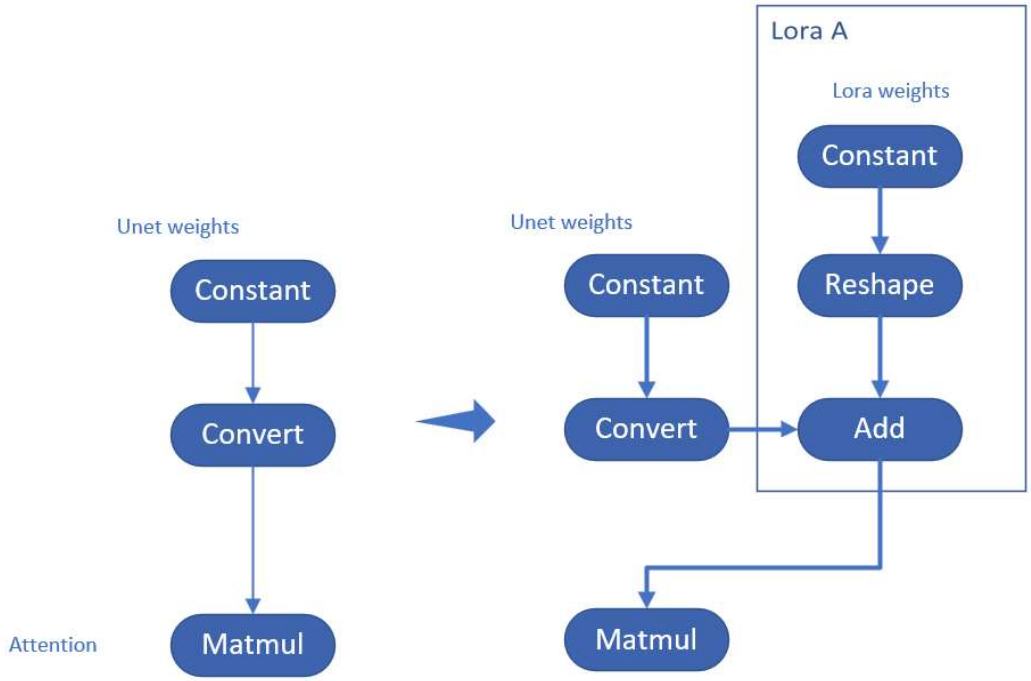
- Workflow:

3. Calculate LoRA weights value:

LoRA weights appending fomular:

$$w_{lora} = \boldsymbol{\alpha} * \boldsymbol{torch.mm(pair\_up, pair\_down)}$$
$$w = w_0 + w_{lora}$$

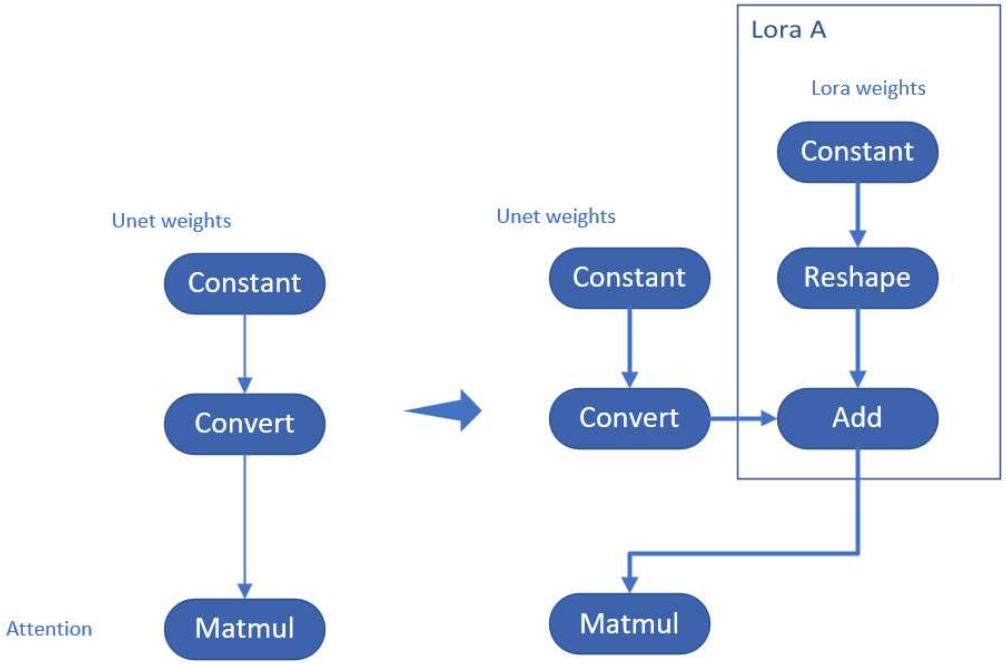
Enable LoRA on OpenVINO™ model in runtime



- Workflow:

4. Create function to insert subgraph into OV model by openvino.runtime.passes API
   - Call manager.register_pass() to register subgraph insertion.
   - Call manager.run_pass() to insert subgraph

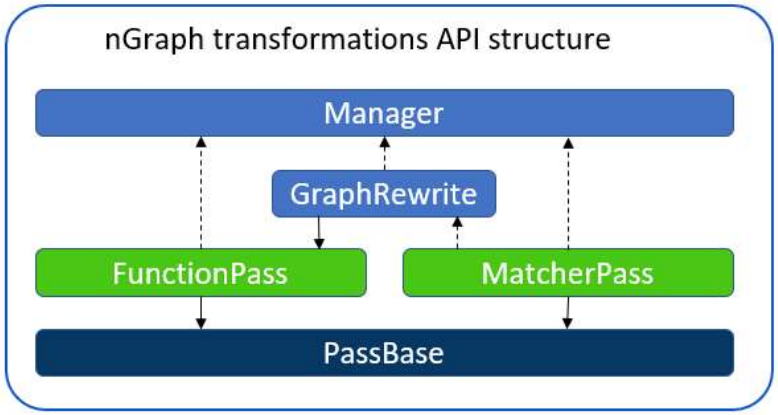
Enable LoRA on OpenVINO™ model in runtime



- Workflow:

5. Compile graph changed Unet/text_encoder OV models

Enable LoRA on OpenVINO™ model in runtime



Ngraph transformation API will do constant folding during model compiling to combine 'Add' and 'Matmul' function together.
Thus, the subgraph insert will not bring performance drop in runtime.

Enable LoRA on OpenVINO™ model in runtime

```python
class InsertLoRA(MatcherPass):
    def __init__(self,lora_dict_list):
        MatcherPass.__init__(self)
        self.model_changed = False

        param = WrapType("opset10.Convert")

        def callback(matcher: Matcher) -> bool:
            root = matcher.get_match_root()
            if root is None:
                return False
            root_output = matcher.get_match_value()
            for y in lora_dict_list:
                root_name = root.get_friendly_name().replace('.','_')
                if root_name.find(y["name"]) != -1 :
                    consumers = root_output.get_target_inputs()
                    lora_weights = ops.constant(y["value"],Type.f32,name=y["name"])
                    reshaped_lora_w = ops.reshape(lora_weights,root_output.get_shape(),special_zero=True)
                    add_lora = ops.add(root,reshaped_lora_w,auto_broadcast='numpy')
                    for consumer in consumers:
                        consumer.replace_source_output(add_lora.output(0))

                    # For testing purpose
                    self.model_changed = True
                    # Use new operation for additional matching
                    self.register_new_node(add_lora)
                    lora_dict_list.remove(y)

            return True

        self.register_matcher(Matcher(param,"InsertLoRA"), callback)
```

- Use openvino.runtime.passes API to create MatcherPass function to insert subgraph into model.
- MacherPass function can only filter layer by type; The logic is:
  - Filter out all convert layer;
  - Check if layer name exist in LoRA weights list;
  - Create subgraph by OV opset.
  - Insert into model
  - Loop until finish all LoRA weights insertion.
- Callback function process new node registration concurrence at backend.

Enable multiple LoRA on OpenVINO™ model in runtime

- Enable multiple LoRA weights into SD model

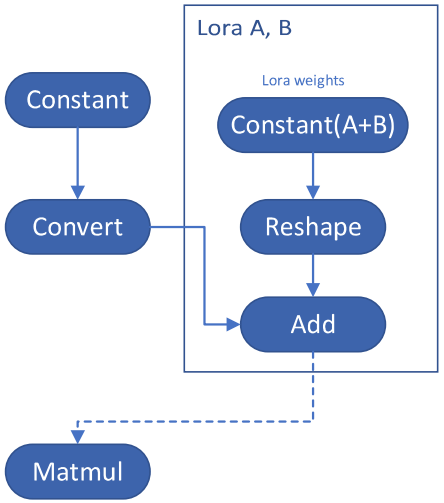
$$w_{loraA} = \alpha_A * torch.mm(pair\_up_A, pair\_down_A)$$

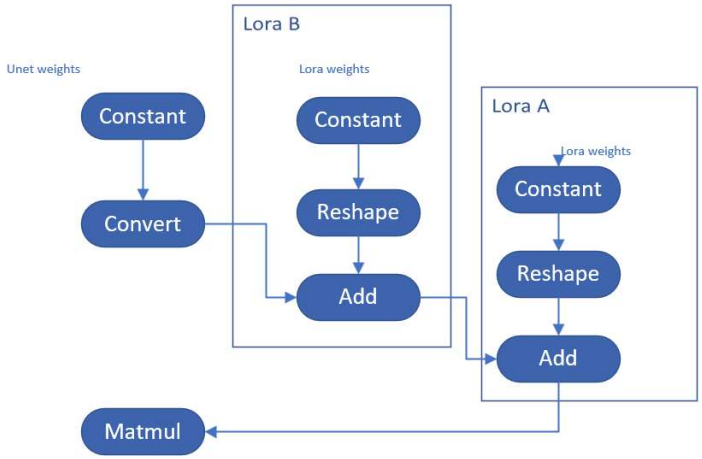
$$w_{loraB} = \alpha_B * torch.mm\left(pair_{up_B}, pair_{down_B}\right)$$

$$w_{lora} = w_{loraA} + w_{loraB}$$

$$w = w_0 + w_{lora}$$

Enable multiple LoRA on OpenVINO™ model in runtime





- Append multiple LoRA weights, compile model once

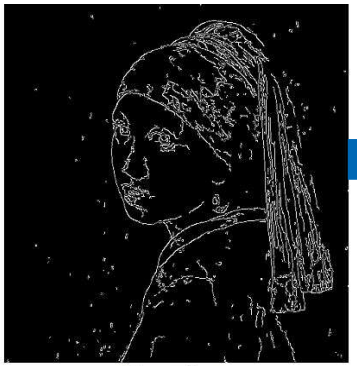
- Cost when inserting subgraph twice
  - Subgraph insertion with manger.run_passes() function
  - Complicated model graph when adding more LoRA weights.

Enable A1111 themed LoRA on OV model in runtime

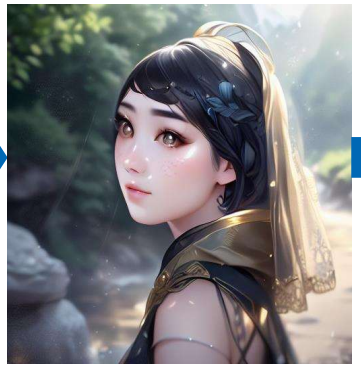
- Create a demo of SD+ controlNet-canny+ 2*LoRA:

  Dreamlike-anime-1.0 SD v1.5 + ControlNet canny + LoRA noiseoffset + LoRA soulcard

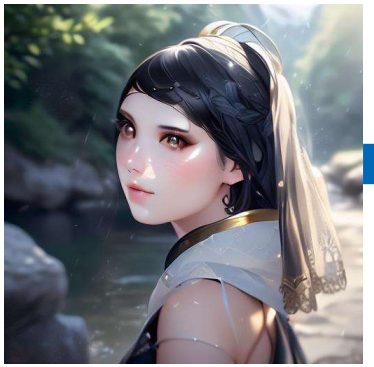
input image



Canny image



dreamlike-anime-1.0 SD + controlnet



SD + controlnet + LoRA noiseoffset

SD + controlnet + LoRA noiseoffset + LoRA soulcard

Enable A1111 themed LoRA on OV model in runtime

Python source code

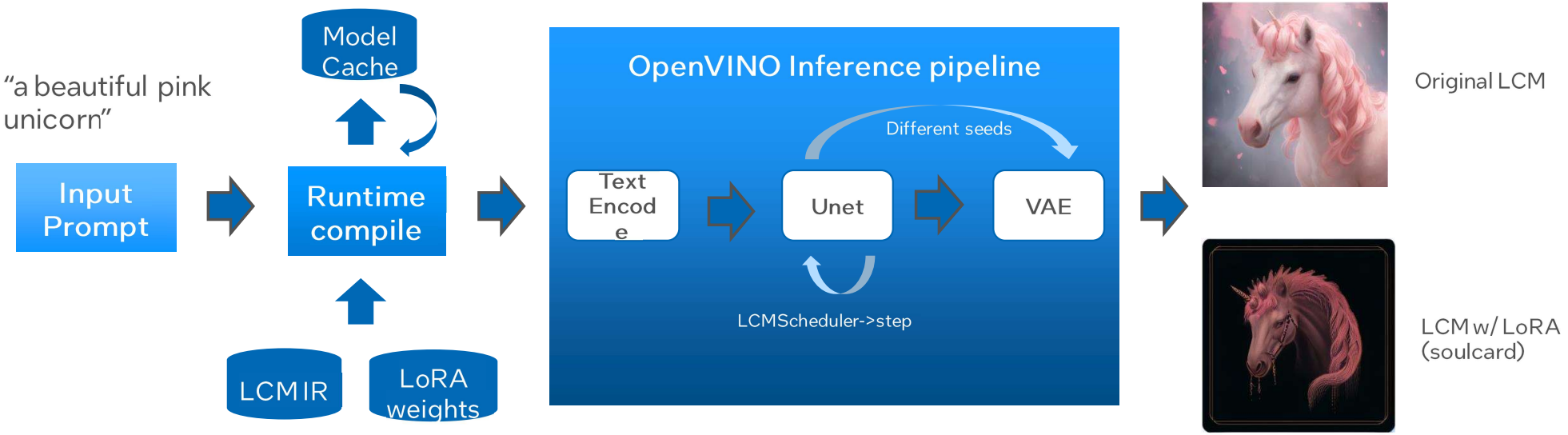
CPP source code

Technical blog

Practical part

Hands-on practice

- LCM with LoRA pipeline (C++) by OpenVINO™ native API



https://github.com/openvinotoolkit/openvino.genai/tree/master/image_generation/lcm_dreamshaper_v7/cpp

Hands-on practice

- LoRA weights enablement
  - Runtime modify model graph to insert operation by adding LoRA weights bias
  - Support runtime model graph modification by OpenVINO™ pass API Matcher interface
  - Enhance LoRA enablement loop of each layer by callback function

```
InsertLoRA::InsertLoRA(LoRAMap& lora_map) :
    m_lora_map(&lora_map) {
    OPENVINO_ASSERT(!m_lora_map->empty(), "Map with LoRA weights is empty");

    auto pattern = ov::pass::pattern::wrap_type<ov::op::v0::MatMul, ov::op::v1::Convolution>();

    ov::matcher_pass_callback callback = [=](ov::pass::pattern::Matcher& m) {
        auto root = m.get_match_root();
        if (!root) {
            return false;
        }
        std::string root_name = root->get_friendly_name();
        std::replace(root_name.begin(), root_name.end(), '.', '_');

        auto it = m_lora_map->begin();
        while (it != m_lora_map->end()) {
            if (root_name.find(it->first) != std::string::npos) {
                ov::Output<ov::Node> weights_port = root->input_value(1);
                std::set<ov::Input<ov::Node>> consumers = weights_port.get_target_inputs();
                auto reshaped_const = std::make_shared<ov::op::v0::Constant>(*(it->second), weights_port.get_shape());
                auto lora_add = std::make_shared<ov::op::v1::Add>(weights_port, reshaped_const);
                for (auto consumer : consumers) {
                    consumer.replace_source_output(lora_add->output(0));
                }
                register_new_node(lora_add);
                it = m_lora_map->erase(it);
                break;
            } else {
                it++;
            }
        }
        return true;
    };

    // Register pattern with Parameter operation as a pattern root node
    auto m = std::make_shared<ov::pass::pattern::Matcher>(pattern, "InsertLoRA");
    // Register Matcher
    register_matcher(m, callback);
}
```

Hands-on practice

- Apply LoRA during model runtime compiling
  - Support Alpha % of LoRA effects
  - LoRA weights bias for text_encoder
  - LoRA weights bias for Unet
- Can scale to support multiple LoRA with a list of LoRA weights files

```
StableDiffusionModels compile_models(const std::string& model_path, const std::string& device,
                                     const std::string& lora_path, const float alpha, const bool use_cache) {
    StableDiffusionModels models;

    ov::Core core;
    if (use_cache)
        core.set_property(ov::cache_dir("./cache_dir"));
    core.add_extension(TOKENIZERS_LIBRARY_PATH);

    // read LoRA weights
    std::map<std::string, InsertLoRA::LoRAMap> lora_weights;
    if (!lora_path.empty()) {
        lora_weights = read_lora_adapters(lora_path, alpha);
    }

    // Text encoder
    {
        auto text_encoder_model = core.read_model(model_path + "/text_encoder/openvino_model.xml");
        apply_lora(text_encoder_model, lora_weights["text_encoder"]);
        models.text_encoder = core.compile_model(text_encoder_model, device);
    }

    // UNet
    {
        auto unet_model = core.read_model(model_path + "/unet/openvino_model.xml");
        apply_lora(unet_model, lora_weights["unet"]);
        models.unet = core.compile_model(unet_model, device);
    }

    // VAE decoder
    {
        auto vae_decoder_model = core.read_model(model_path + "/vae_decoder/openvino_model.xml");
        ov::preprocess::PrePostProcessor ppp(vae_decoder_model);
        ppp.output().model().set_layout("NCHW");
        ppp.output().tensor().set_layout("NHWC");
        models.vae_decoder = core.compile_model(vae_decoder_model = ppp.build(), device);
    }

    // Tokenizer
    {
        // Tokenizer model wil be loaded to CPU: OpenVINO Tokenizers can be inferred on a CPU device only.
        models.tokenizer = core.compile_model(model_path + "/tokenizer/openvino_tokenizer.xml", "CPU");
    }

    return models;
```

Hands-on practice

```
# Step 1: Prepare build environment
$ conda create -n openvino_lcm_cpp python==3.10
$ conda activate openvino_lcm_cpp
$ conda update -c conda-forge --all
$ conda install -c conda-forge openvino=2024.1.0 c-compiler cxx-compiler git make cmake
# Ensure that Conda standard libraries are used
$ conda env config vars set LD_LIBRARY_PATH=$CONDA_PREFIX/lib:$LD_LIBRARY_PATH

# Step 2: LCM and tokenizer model export
$ git submodule update --init
$ conda activate openvino_lcm_cpp
$ python -m pip install -r requirements.txt
$ python -m pip install ../../../thirdparty/openvino_tokenizers/[transformers]
$ optimum-cli export openvino --model SimianLuo/LCM_Dreamshaper_v7 --weight-format fp16
models/lcm_dreamshaper_v7/FP16

# Step 3: Build the LCM application
$ conda activate openvino_lcm_cpp
$ cmake -DCMAKE_BUILD_TYPE=Release -S . -B build
$ cmake --build build --config Release -parallel

# Step 4: Run pipeline
$ ./build/lcm_dreamshaper [-p <posPrompt>] [-s <seed>] [--height <output image>] [--width <output image>]
[-d <device>] [-r <readNPLatent>] [-a <alpha>] [-h <help>] [-m <modelPath>] [-t <modelType>]
```

Conclusion

Conclusion

- Motivation of LoRA for text-to-image
- Enable LoRA weights on OpenVINO™ models offline
- Enable LoRA weights on OpenVINO™ models in Runtime
  - Single and Multiple LoRA weights
- Hands-on practice
  - Set up an LCM pipeline in C++ with OpenVINO™
  - Adjust alpha % of LoRA effects, LoRA weights bias for text_encoder, LoRA weights bias for Unet


intel