Alg Coding Test 1: ***Predicting scene meshes from depth estimation***

- **Background:** Mesh is the commonly used geometry representation in 3D intelligence. However, accurate mesh requires specific sensors to be obtained, e.g., Lidar or RGBD cameras. It is worth noting that a lot of AR/VR devices do not have such sensors, e.g., Meta Quest 2 and Quest Pro. Thus how to get the mesh of the captured scene from RGB images is an important problem to solve. There are already a lot of end-to-end deep-learning-based methods, e.g., NeuralRecon, but they are usually overfitted to a specific dataset. In this project, let's dive into a less accurate but more general way to get the mesh, i.e., reconstructing mesh from deep-learning-based depth estimation.

- **Problem Formulation:** To get the mesh from a deep-learning-based depth estimation model, we need to:
    - Step 1: Find a depth estimation model that takes the RGB image as the input and outputs the corresponding depth. Here, let's use DistDepth, which can provide the absolute depth from the origin of the cameras to the surface of the objects in the scene.
    - Step 2: Merge the depth estimation results from multiple images into one point cloud. Please refer to this script to learn how to convert the depth estimation to the point cloud based on the intrinsic and extrinsic of the camera.
    - Step 3: Convert the point cloud into mesh by first converting the point cloud into a 3D grid (you can define the resolution of the 3D grid yourself, but we recommend 256*256*256), where each item in the 3D grid represents the normalized number of points (normalized to the max value of all items), which is in the range of [0, 1]. Then, we convert such a 3D grid into mesh via a marching cube. The marching cube implementation is described in this script.
    - Step 4: Fine-tune the mesh's vertices positions and colors using nvdiffrec with the RGB images as training data. To simplify the fine-tuning process, please just learn the RGB color and position for each vertex instead of the combination of PBR materials and environment light in the official codebase of nvdiffrec.

- **Implementation Details**
    - **Dataset**: scene_0241 of ScanNet**.** To reduce the runtime, you can (1) subsample the dataset by selecting one frame from every five frames of the images to be depth extracted and (2) resize each image to 640*480 and central crop it to 624*468 to avoid the camera distortion at the edges of the image. You can download scene_0241 following the instructions here.
    - **Framework:** You can use any deep learning frameworks and coding languages you want, but please note that DistDepth and nvdiffrec are in PyTorch.
    - **Expected Results:** Please provide the following four files.
        - The point cloud from the resized first image in scene_0241, saved in .ply
        - The mesh from all the subsampled and resized images in scene_0241, saved in .ply. Please note that you may need to tune the level of the marching cube to find the best-reconstructed mesh.

        - The fine-tuned mesh with the RGB color assigned to each vertice. You can tune the training iteration and batch size for better rendering quality. But the default values in the official codebase of nvdiffrec (i.e., batch size = 1 and iteration = 5000) should be sufficient to generate better results as compared to the mesh before fine-tuning.
        - A report to include (1) how you get the final reconstructed mesh step by step, (2) the quality of your reconstructed mesh as compared to the provided mesh in the ScanNet dataset, which is from an RGBD camera, (3) the quality of your fine-tuned mesh as compared to the provided mesh in the ScanNet dataset and the mesh before fine-tuning, and (4) your ideas on how to further improve this mesh reconstruction pipeline.

Alg Coding Test 2: ***Efficient LLMs via Switchable and Dynamic Quantization***

- **Background:** Large language models (LLMs) have garnered increasing attention due to their remarkable emergent abilities, albeit at the expense of their substantial model size. However, this large size also poses challenges in terms of efficiency. Among various efforts to enhance LLM efficiency, quantization has emerged as a promising approach, offering a favorable accuracy-efficiency trade-off, ease of implementation, and compatibility with hardware. In this project, our goal is to further investigate this method by developing a switchable and dynamic quantization scheme that aims to improve the accuracy-efficiency trade-off of LLMs.

- **Problem breakdown:** To achieve the goal of switchable and dynamic quantization, you need to finish the following steps:
    - Step 1: Integrate quantization into the pretrained GPT-2 model. Given the potentially diverse redundancy in each layer of the model, implement the quantization function to support using different bit-width per layer based on the input config.
    - Step 2: Add multiple LoRA modules to all linear layers in GPT-2 and implement the function to adaptively activate different LoRA modules in each layer during inference based on the input config.
    - Step 3: Tune the GPT-2 model for 1000 iterations on the SQuAD dataset with different per-layer quantization bit-width configurations at the same time (i.e., enabling switchable precision), where different LoRA modules are activated for different per-layer quantization bit-width configurations.
    - Step 4: Evaluate the performance of your tuned model under different quantization bit-width configurations on SQuAD dataset. You may decide your own quantization configuration, i.e., the optimal per-layer bit-width configuration to push the accuracy-efficiency trade-off. Try to draw some insights based on your observations.
    - Step 5: Instead of jointly training with all bit-widths in Step 3, use the cyclic precision training to dynamically change the training bit-widths throughout the tuning process and investigate whether this quantization strategy, which effectively

enhances the accuracy of CNNs, can also improve the downstream accuracy of LLMs.
    - Step 6: On top of a pretrained GPT-2, examine whether random precision switch, i.e., dynamic quantization at inference time, can improve GPT-2's adversarial robustness. You can do a literature survey and pick arbitrary widely used adversarial attacks within 3 years to evaluate GPT-2's adversarial robustness.

- **Implementation Details**
    - *Codebase references*
        - Model: GPT-2
        - Quantization scheme: QAT-LLM
        - Downstream dataset: SQuAD
    - *Deliverables*: Submit your code with a brief ReadMe and a PDF report to answer the following questions:
        - [Step 4] What is the task accuracy achieved after applying various quantization bit-width configurations to the SQuAD dataset?
        - [Step 4] How did you determine the optimal quantization bit-width configurations? Have you gleaned any insights from your observations that could guide future work to further enhance performance?
        - [Step 4] A motivation behind switchable quantization is to support diverse layer-wise quantization configurations simultaneously, accommodating different resource allocation needs. Could you suggest additional training objectives that could more effectively facilitate the mechanism for switching quantization bit-widths?
        - [Step 5] Does this phenomenon align with the observations in CPT (ICLR'21)? If not, what could be the potential reasons?
        - [Step 6] Does this phenomenon align with the observations in Double-Win Quant (ICML'21)? If not, what could be the potential reasons?
        - Based on your explorations of switchable and dynamic quantization, could you propose some promising research directions or questions for further integrating them with LLMs?

Alg Coding Test 3: Deploying Efficient Deep Neural Networks on Mobile Devices

- **Background:** To run Deep Neural Networks (DNNs) on mobile or IoT devices, many efficient DNNs are manually designed or automatically searched to accommodate the limited computing resources of these devices. Among these efficient DNNs, FBNet is one of the classical networks searched by Neural Architecture Search automatically. However, in the official codebase of FBNet, there is no instruction on how to deploy the pre-trained FBNet on mobile devices. Thanks to the powerful deployment tools provided by TF Lite, the deployment will be straightforward once we have the pre-trained model in tflite format. In this project, let's dive into the process of converting PyTorch pre-trained models to tfLite for deployment on mobile devices.
- **Problem Formulation:** To get the tflite model for mobile devices from a pre-trained PyTorch model, we need to:

- Step 1: Load the pre-trained PyTorch model from the official FBNet codebase.
- Step 2: Convert the PyTorch model into a tflite compatible model format (e.g., TensorFlow or Keras). Generally, there are two options for doing this.
  - Option 1: Rewrite the model architecture in TensorFlow or Keras and copy the weights from PyTorch with the help of other Python libraries (e.g., NumPy).
  - Option 2: Use existing model conversion intermediate format libraries (e.g., ONNX).

  For Option 1, the risk is that the accuracy may decrease if there are mistakes in weight copying and pasting. For Option 2, the risk is that the conversion may not always succeed and may result in some unnecessary (e.g., reshape to accommodate NCHW and NHWC format differences) or unsupported operators for tflite.
- Step 3: Verify the accuracy of the converted model in tflite compatible model format. You do not need any dataset to do the verification. Instead, you can provide a set of random inputs to the models in both PyTorch format and tflite compatible model format, and compute the Mean Squared Error (MSE) between the outputs of the two models. (1 - MSE) can be used as the accuracy metric.
- Step 4: Convert the model in tflite compatible model format to the tflite format and check the accuracy again using tflite_runtime on you local devices. Apply 8 bit (8 bit weights and 8-bit activations) post-training quantization to the tflite model, and check the accuracy again.

- **Implementation Details**
  - **Models:** fbnet_a and fbnet_b
  - **Expected Results:**
    - The models in tflite compatible model format
    - The unquantized models in tflite format
    - The quantized models in tflite format
    - A report to include (1) how you convert the models in PyTorch format to tflite format, (2) the accuracies of models in PyTorch format, models in tflite compatible model format, unquantized models in tflite format, and quantized models in tflite format, and (3) your ideas on how to further improve the model conversion and deployment pipeline, especially when you have a large number of models to convert.

Alg Coding Test 4: Understanding Deep Neural Networks Through Spline Approximation

- **Background**: Recent advancements in Deep Neural Networks (DNNs) have been impressive, yet the underlying mechanisms behind their exceptional performance remain largely unexplored. A promising area of research is the use of spline approximation theory to shed light on these mechanisms. In this context, each activation node in a DNN can be seen as a subdivision line within the network's input space. This approach divides the DNN input space into distinct, non-overlapping regions, each contributing to the network's overall input-output behavior. As shown in Fig. 1, the partition Ω of the DNN input space is composed of non-empty and non-intersecting regions ω that collectively represent the entire DNN input space, leading to the following DNN input-output form. You can also refer to the associated research paper Mad Max for more detailed insights.

$$f(\boldsymbol{x}) = \sum_{\omega \in \Omega} (A_\omega \boldsymbol{x} + \boldsymbol{b}_\omega) \mathbb{1}_{\boldsymbol{x} \in \omega}$$

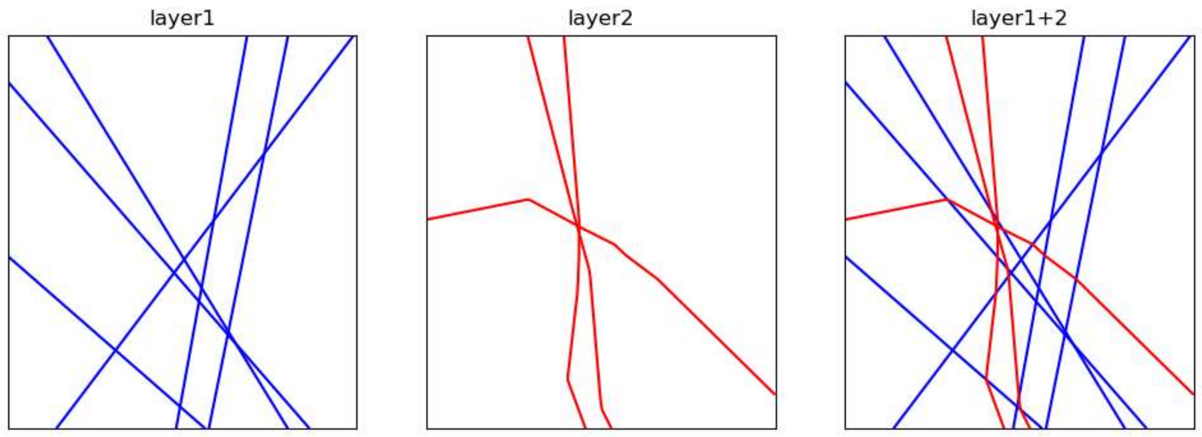

Figure 1:Spline Visualization Example

- **Problem Formulation**: We expect you to utilize these codes to create visualizations during the training of DNNs or pruned DNNs. To aid you in this exploration, we have prepared code examples (accessible via Google Drive) to assist in visualizing these splines. Your tasks are as follows:
    - **Fully-Connected Networks (FCNets)**: Visualize the splines throughout the training process of two layer FCNets. Then, try to prune some hidden nodes or weights with various pruning criteria/granularities/ratios and observe the resulting changes in the training trajectory of subdivision lines.
    - **Convolutional Networks (ConvNets)**: Implement similar visualizations for ConvNets along the training trajectories, focusing on different pruning scenarios (e.g., 20%, 40%, 60%, and 80%). Try to make it clear by considering a small net or select few subdivision lines.
    - **Report Submission**: Compile your findings, insights, and learning outcomes into a comprehensive report. Please submit this report to Professor Yingyan Lin.

- **Implementation Details**
    - **Model:** 2-layer FCNets or ConvNets
    - **Pruning scheme**: Magnitude pruning, lottery ticket pruning, etc
    - **Downstream dataset**:
        - Simple one: Handcrafted binary classification in 2D input space
        - Extended one: MNIST
    - **Guide Questions:** Answer the following questions
        - **Dependency Analysis**: What dependencies exist between subdivision lines across different layers of the network?

        - **Pruning and Subdivision Relationship**: How does pruning affect the subdivision lines? What relationships can you infer between the two?
        - **Training Trajectory Observations**: What observations or findings can you gather from visualizing the training trajectory? Based on your insights, how can one determine the most effective pruning strategy?

HW Coding Test 1: ***4x4 Systolic Array Design & Evaluation***

- **Overview:** Design a 4x4 Systolic Array in Verilog, write the testbench and evaluate the PPA (power, performance, and area)
- **Requirements:** Your design should contain the following parts: a systolic array that supports fix-point 16-bit multiplication & accumulation, two memory blocks that feed the data into the array, a controller that controls the memory and the systolic array, one memory block to store the output data, and one memory block to store the instruction.
    - The top IO of the design should be
        - clk, rst (rst_n if targeting ASIC)
        - addrA, enA, dataA (for write data memory A)
        - addrB, enB, dataB (for write data memory B)
        - addrI, enI, dataI (for write instruction memory I)
        - addrO, dataO (for read result memory O)
        - ap_start (pulse signal)
        - ap_done (level signal)
    - Instruction is an integer, which marks the size of the matrix. If the instruction is 0, it means it is the end of the execution. For example, if you get the following instruction.
        - [4, 8, 16]
        - It means that you will need to perform 4x4 * 4x4 Matrix-Matrix Multiplication, then an 8x4 * 4x8 MMM, and finally a 16x4 * 4x16 MMM.
    - The data should be directly stored in the data memory. During the execution, there should be no data transfer between the module and other modules.
    - For the data transfer between the memory and array, both the input and output data should be systolically propagated, i.e., you can not use a large mux just to pick which PE among all the PEs to send the data directly to.
    - You should design your own test data generator. One recommended way is to use MATLAB to generate random matrices and use the “fi” command to generate a fixed representation.
- **Testbench guidelines:** In the Verilog testbench, you should first write data to the data memory and the instruction memory. You should at least test the [4, 8, 16] in the instruction memory. After the input data and memory are loaded, use ap_start to start your design. After getting the ap_done signal, print the output data from the memory to a file and compare the results.
- **Bonus**: Design an FP16 PE and do the flow again.
- **Deliverable:** After the task, the source code and the documentation, which have enough information to explain the results and implementation flow, should be provided for assessment.

Design source:
- Python / C / MATLAB file to generate fixed point inputs
- Verilog design & testbench
- Result evaluation program

Documentation

HW Coding Test 2: ***SOTA AI accelerator arch-level reproduction***

- **Overview:** This test involves reproducing a state-of-the-art AI accelerator using Verilog, aiming to achieve the closest possible resemblance.
- **Requirements:**
  - **Design Approach:**
    - Adopt a top-down approach in designing the system architecture.
    - Start with a block diagram to outline the system and define specifications for each module.
    - Proceed with detailed implementation for each module, before integrating them into the complete system.
  - **Technical Specifications:**
    - Use Hardware Description Languages (HDL) such as Verilog or SystemVerilog, or opt for hardware performance modeling using C/Python.
    - Your design must include the following:
      - Algorithm compilation that involves mapping and scheduling tasks on the hardware.
      - RTL design or performance modeling.
      - Testing and evaluation of the design.
  - **Project Scope:**
    - Completion of the project before the meeting is not required.
    - Emphasis is on your thought process, architectural approach, and future completion plans.
  - **Extendability requirements:**
    - If we were to use the design to profile some customized workloads, e.g., another transformer model not benchmarked by the work, we would have the ways to know the hardware performance
    - If we were to make modifications to the original hardware in the paper, we could base the modifications on your design
- **Choices of AI Accelerators for Reproduction:**
  - SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning
  - SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training
- **Deliverable:**
  - **Source Code and Documentation:**
    - All source files for design, testing, simulation, and evaluation.
    - Comprehensive documentation that includes:

- Detailed implementation plan and process.
- Instructions for potential users on how to run or modify the design.
- Clear delineation of completed and pending tasks.
- A roadmap for future work to complete the design.