ADVANCE COMPUTER ARCHITECHTURE
CS G524

BITS Pilani
K K Birla Goa Campus

PROJECT MILESTONE 1

To Dr. Kanchan Manna

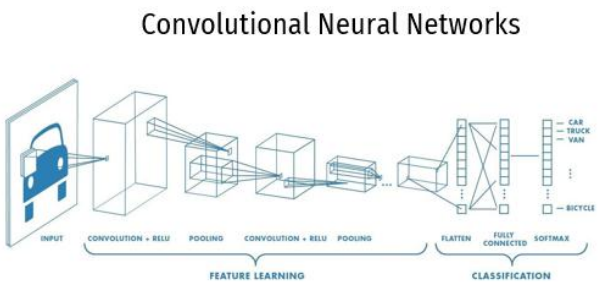
From Sai Charan, Aditya Agarwal, Yogesh Agrawal

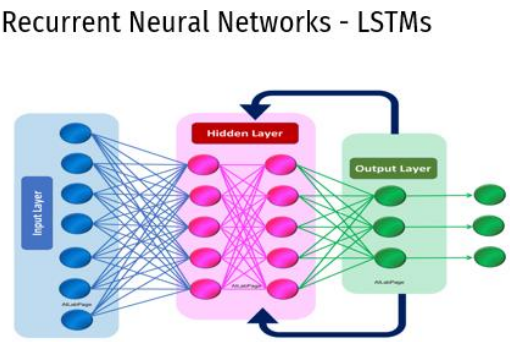
Date Apr 14, 2025

Subject: Milestone 1 report – Branch prediction algorithms

1. Introduction / Project Overview

This project investigates advanced branch prediction by integrating Reinforcement Learning (RL) and Neural Network (NN) techniques. Our goal is to evaluate and potentially improve predictor accuracy beyond traditional methods. We leverage concepts from the "Branch Prediction as an RL Problem" paper [4] and implementations from the "Branch Prediction Using ANN" thesis/repository [3], using the ChampSim simulator [1] for evaluation.





2. Milestone 1 Objectives

- Set up the software environment (Python, PyTorch, ChampSim, project code).
- Generate branch prediction datasets, including Hard-to-Predict (H2P) branch identification for one benchmark (541.leela).
- Train baseline NN models (BranchNet, Conv-LSTM) from the repository [3] on the generated dataset.
- Perform initial evaluation of trained models on H2P branches (accuracy/MPKI).
- Familiarize with the codebase, data workflow, and reference materials [3, 4].

3. Work Completed in Milestone 1

- Environment Setup: Successfully configured the Python environment, installed necessary libraries (PyTorch, etc.), obtained and set up the modified ChampSim simulator, cloned the project repository [3], and configured paths.
- Dataset Generation (541.leela): Obtained SPEC CPU 2017 traces for 541.leela. Used ChampSim to identify H2P branches based on frequency and misprediction criteria. Extracted relevant branch histories and utilized the create_ml_datasets.py script to generate h5py datasets suitable for training.
- Model Training (541.leela): Employed branchnetTrainer.py to train separate BranchNet (using big.yaml) and Convolutional-LSTM models for each identified H2P branch of the 541.leela benchmark, using the generated h5py data.
- Model Evaluation (541.leela): Used BatchDict_testerAll.py to evaluate the trained models. The script processed a reference trace, loaded the appropriate .pt model for each encountered H2P branch, performed batched predictions, calculated accuracy statistics, and generated prediction trace files.
- **Preliminary Results**: Generated accuracy metrics for BranchNet and Conv-LSTM on 541.leela H2P branches, confirming the models' ability to learn predictive patterns. Quantitative comparisons are pending further analysis.
    - Both methods show high accuracy results, reaching close to 99% accuracy.

4. Key Methodology Highlights

H5PY Dataset Creation: Leveraged create_ml_datasets.py to convert raw ChampSim branch history traces into efficient HDF5 files for training data management.

Identifying H2P branches:

- Viewed at least 15,000 times
- They cause 1000 misspredicts

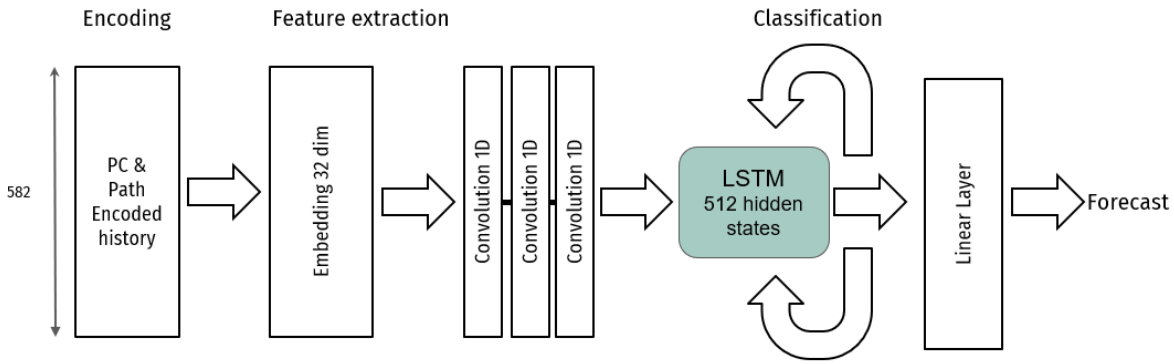
- <99% accuracy with TAGE-SC-L 8KB predictor
- They are common in at least 40% of the available inputs

Model Definition : Implemented models like the one below combining convolutional layers for feature extraction from branch history and LSTM layers for sequence processing.

```
# Simplified structure based on thesis description / models.py RNNLayer
with LSTM
class CNN_LSTM(nn.Module):
    def __init__(self, input_dim, out_dim, num_layers, pc_hash_bits=11,
history_length=582):
        super().__init__()
        self.embedding1 = nn.Embedding(2**pc_hash_bits+1, input_dim)
        # Conv Layers
        self.conv1  =  nn.Conv1d(in_channels=input_dim,  out_channels=64,
kernel_size=1)
        self.relu1 = nn.ReLU()
        # ... more conv layers ...
        self.conv_final           =           nn.Conv1d(in_channels=...,
out_channels=out_dim, kernel_size=1)
        self.relu_conv_final = nn.ReLU()
        # LSTM Layer
        self.rnn    =    nn.LSTM(input_size=out_dim,    hidden_size=512,
num_layers=num_layers, batch_first=True)
        # Final Linear Layers
        self.fc1 = nn.Linear(512, 128)
        self.relu_fc = nn.ReLU()
        self.fc2 = nn.Linear(128, 1)
        self.finalActivation = nn.Tanh()

    def forward(self, seq):
        x = self.embedding1(seq.long()).transpose(1, 2) # Embed and shape
for Conv1D
        # Apply Conv layers ...
        x  =  self.relu_conv_final(self.conv_final(x)).transpose(1,  2)  #
Shape for LSTM
        lstm_out, _ = self.rnn(x)
        last_hidden_state = lstm_out[:, -1, :]
        # Apply Linear layers ...
        x = self.relu_fc(self.fc1(last_hidden_state))
        x = self.fc2(x)
        return self.finalActivation(x).squeeze(dim=1)
```

Original authors approach for conv-LSTM model:



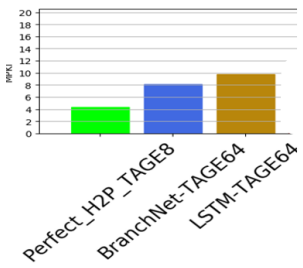
Training Loop Snippet (from branchnetTrainer.py): Standard PyTorch training loop involving forward pass, loss calculation (MSELoss), backpropagation, and optimizer step.

```
# Simplified training step
optimizer.zero_grad()
outputs = model(X.long())
loss = criterion(outputs, labels.float())
loss.backward()
optimizer.step()
```

Evaluation Batching Snippet (from BatchDict_testerAll.py): Efficiently evaluates models by processing trace history, batching inputs for H2P branches, performing GPU predictions, and aggregating results.

```
# Simplified evaluation concept
# ... inside trace loop ...
if ip in modelsDict: # If H2P branch
    # Add history to batchDict[ip]['batch']
    batchDict[ip]['batchCounter'] += 1
    if batchDict[ip]['batchCounter'] == batchSize:
        # Predict using modelsDict[ip] on the full batch
        # Record results, reset counter
        batchDict[ip]['batchCounter'] = 0
# Update history deque
# ... handle remaining batches at end ...
```

The mispredictions per kilo instructions were found to be:



5. Challenges Encountered

- ChampSim Integration: Adapting ChampSim for specific data logging required C++ familiarity.
- Dataset Management: Large trace and h5py file sizes demand significant storage and processing time.
- Training Duration: Training models for numerous H2P branches is computationally intensive.
- Script Complexity: The evaluation script (BatchDict_testerAll.py) required careful analysis due to its batching and output logic.

6. Milestone 2 Objectives / Next Steps

- Implement RL Predictors: Develop G-QLAg and PolGAg based on paper [4], potentially adapting ChampSim or using a separate framework.
- Evaluate Additional NN Models: Train/evaluate Tarsa, CNN, and Transformer models from [3] on 541.leela.
- Cross-Benchmark Evaluation: Test promising NN and RL models on other SPEC benchmarks (505.mcf, 531.deepsjeng, 557.xz).
- Comparative Analysis: Analyze MPKI/IPC trade-offs across all models and benchmarks.
- H2P Branch Investigation: Study characteristics of H2P branches.
- Hybrid Predictor Exploration: Design and potentially test a simple hybrid (RL + NN) predictor in ChampSim.
- Misprediction Penalty Exploration: Investigate modeling penalties within the RL framework or via simulation analysis.

8. References

[1] Quangmire/ChampSim: ChampSim repository. (URL: e.g., https://github.com/ChampSim/ChampSim)

[2] Tarsa, S. J., Lin, C. K., Keskin, G., Chinya, G., & Wang, H. (2019). Improving Branch Prediction By Modeling Global History with Convolutional Neural Networks. *arXiv preprint arXiv:1909.03891*. (Implicitly referenced by thesis)

[3] Vontzalidis, A. (2021). *Branch Prediction using artificial neural networks* [Diploma Thesis]. National Technical University of Athens. GitHub: https://github.com/aristotelis96/thesis-branch-prediciton-ml

[4] Zouzias, A., Kalaitzidis, K., & Grot, B. (2021). Branch Prediction as a Reinforcement Learning Problem: Why, How and Case Studies. *ML for Computer Architecture and Systems 2021*. arXiv:2106.13429.

[5] SPEC CPU 2017 Benchmark Suite. https://www.spec.org/cpu2017/

[6] Paszke, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems 32*.