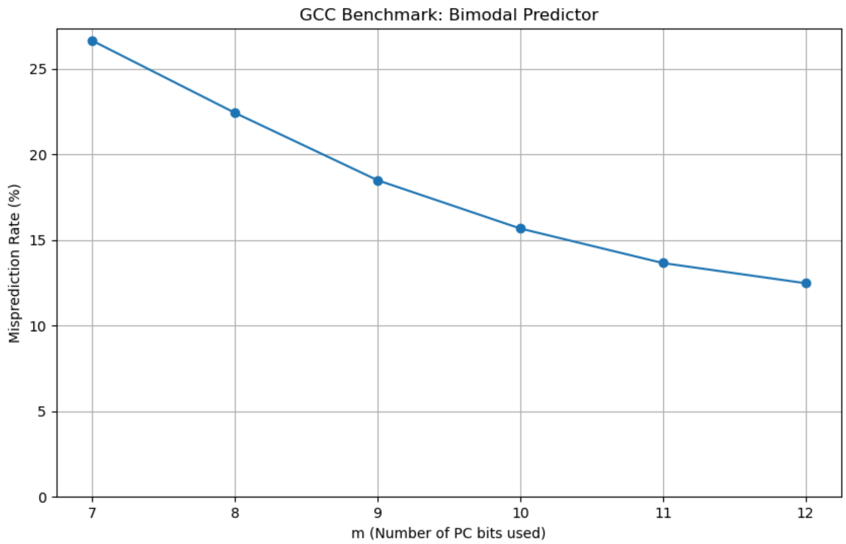
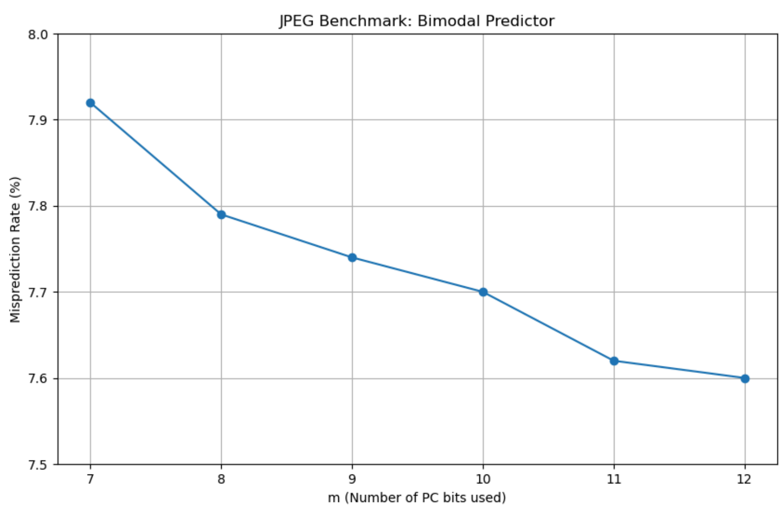
Name: **Varun M**
ROLL No: EE20B149

1. **Bimodal Predictor**
   - **GCC**



   - **JPEG**



Analysis:

GCC Benchmark trend:

- There's a significant improvement in prediction accuracy as m increases, particularly in the lower m values.
- The rate of improvement is higher between m=7 and m=10, with each step providing substantial gains.

- The improvement starts to slow down after m=10, suggesting diminishing returns for larger predictor tables.
- Even at m=12, the misprediction rate is still relatively high at 12.47%, indicating that the GCC benchmark has complex branch behavior that remains challenging to predict accurately.

JPEG Benchmark Trends:

- The overall misprediction rates are much lower compared to GCC, starting at 7.92% for m=7.
- There's only a slight improvement as m increases, with the total reduction being just 0.32 percentage points from m=7 to m=12.
- The rate of improvement is relatively consistent across all m values, without any sharp drops or plateaus.

Similarities between benchmarks:

- Both benchmarks show an overall downward trend in misprediction rates as m increases.
- For both benchmarks, the improvements become less pronounced at higher m values, indicating diminishing returns for larger predictor tables.

Differences:

- Initial misprediction rates: GCC starts with a much higher misprediction rate (26.65% at m=7) compared to JPEG (7.92% at m=7).
- Magnitude of improvement: GCC shows a dramatic improvement, reducing misprediction rate by 14.18 percentage points, while JPEG only improves by 0.32 percentage points.
- Sensitivity to m: GCC's performance is much more sensitive to changes in m, especially for lower m values. JPEG's performance, on the other hand, is only marginally affected by increases in m.
- Final misprediction rates: Even at m=12, GCC's misprediction rate (12.47%) is still significantly higher than JPEG's worst rate (7.92% at m=7).

Overall Observation:

- The GCC benchmark, which represents a compiler workload, seems to have more complex and harder-to-predict branch behavior. This could be due to the diverse code patterns and decision structures typically found in compiler operations.

- The JPEG benchmark, representing image processing, appears to have more predictable branch behavior. This might be due to more regular, loop-based operations common in image processing algorithms.
- For GCC, increasing the predictor size (m) provides substantial benefits, especially up to m=10. This suggests that for workloads similar to compilers, larger branch predictors can significantly improve performance.
- For JPEG, the benefit of increasing predictor size is minimal. This implies that for workloads similar to image processing, even smaller predictors (m=7 or 8) might be sufficient, and allocating more resources to the branch predictor may not be cost-effective.
- If designing a general-purpose processor that needs to handle both types of workloads efficiently, a compromise might be to choose m=10 or m=11. This would provide most of the benefit for GCC-like workloads while not oversizing the predictor for JPEG-like workloads.

Proposed Branch Predictor design:

The maximum allowed m = 16 as the storage for bimodal is 2^m. 2 bits
$2^{m+1} <= 2^{17}$

m <= 16

For Gcc benchmark

As m increases, the misprediction rate decreases significantly, especially from **m = 7** to **m = 11**.

or the **gcc benchmark**, the misprediction rates for different values of mmm are:

- **m = 10**: 15.67%
- **m = 11**: 13.65%
- **m = 12**: 12.47%

While the **misprediction rate** at m=11 is **13.65%**, which is slightly higher than m=12 (12.47%), it represents a significant drop compared to m=10(15.67%). The decrease in misprediction rate from m=10 to m=11 is **2.02%**, while the further improvement from m=11 to m=12 is only **1.18%**.

This indicates that **m = 11** offers a substantial reduction in misprediction rate compared to **m = 10**, while not requiring the additional storage that comes with **m = 12**.

2. Storage Cost

For **m = 11**, the storage requirement is:

$2^{11} \times 2/8 = 512$ bytes This is a reasonable storage requirement, well within the **16kB** budget. Compared to **m = 10** (which uses **256 bytes**), it doubles the storage requirement but provides a meaningful improvement in prediction accuracy.

3. Trade-off Between Accuracy and Cost

The trade-off between storage and accuracy suggests that **m = 11** strikes a balance:

- **m = 10** offers lower storage but a higher misprediction rate.
- **m = 12** has a slightly better misprediction rate but comes at the cost of larger storage (1024 bytes).

Thus, **m = 11** can be considered the **sweet spot**, offering a **significant reduction in misprediction rate** over **m = 10** while avoiding the diminishing returns of increasing to **m = 12**.

Proposed Design for GCC with m = 11:

- **Misprediction rate**: **13.65%**.
- **Storage requirement**: **512 bytes**.

This configuration offers a good balance between prediction accuracy and storage cost, making it a solid choice for the **gcc benchmark**.

Jpeg Benchmark:

The misprediction rates for the **jpeg** benchmark for various values of mmm are:

- **m = 7**: 7.92%
- **m = 8**: 7.79%
- **m = 9**: 7.74%
- **m = 10**: 7.7%
- **m = 11**: 7.62%
- **m = 12**: 7.6%

Analysis:

- The misprediction rate shows very little improvement beyond **m = 8**. The change from **m = 8** to **m = 12** is only **0.19%**.

- Since the improvement in prediction accuracy is minimal after **m = 8**, there's little justification for increasing mmm further.

Storage Consideration:

- For **m = 8**, the storage requirement is: 28×2/8=64 bytes 2^8
- For **m = 9**, the storage requirement is: 2^9×2/8=128 bytes
- For **m = 10**, the storage requirement is: 210×2/8=256 bytes

The predictor for **m = 8** is highly efficient, requiring only **64 bytes**, and further increasing mmm gives marginal improvements in misprediction rate.

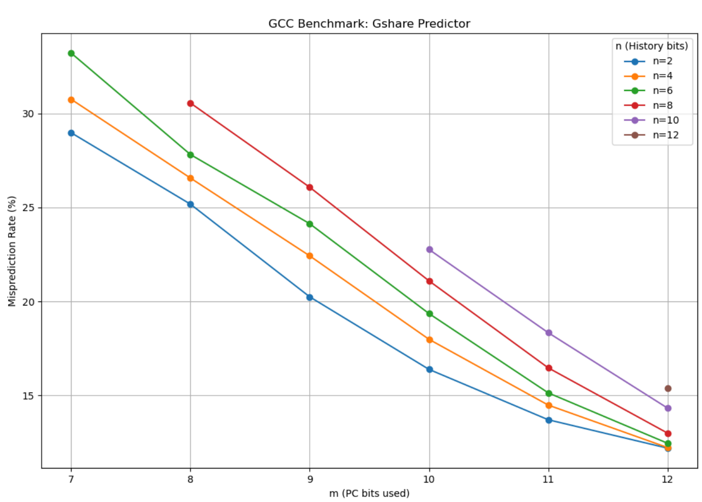
Proposed Design for JPEG:

- **m = 8** is optimal for the **jpeg** benchmark:
    - **Misprediction rate**: 7.79%
    - **Storage requirement**: **64 bytes**.

This configuration provides good prediction accuracy at a minimal storage cost, which leaves ample room within the 16kB budget.

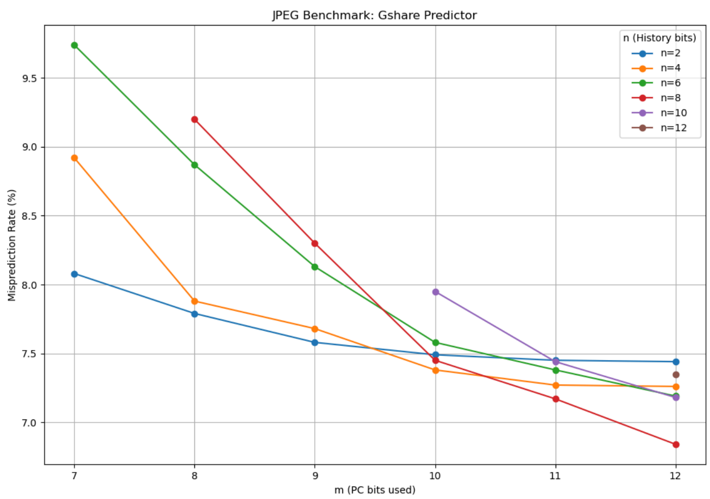
2. Gshare Predictor

GCC



- As m increases for all values of n, the misprediction rate decreases.
- For small values of n (e.g., n=2,4), the misprediction rate starts high but drops significantly as m increases.

- Larger values of n (e.g., n=10,12) show a steeper decrease in misprediction rates for smaller values of mmm, but the benefit diminishes beyond m=10.
- The lowest misprediction rates are observed at high m and high n.
- **JPEG**



- For the **jpeg** benchmark, the misprediction rate is consistently lower than that of the gcc benchmark.
- The trend is similar: increasing m reduces the misprediction rate for each value of n.
- The rates start to stabilize at higher values of m and show diminishing returns in prediction accuracy, especially for n=10 and n=12.
- The misprediction rates for jpeg are relatively stable compared to gcc, with less dramatic improvement as m increases.

ANALYSIS

Trends:

- For **gcc**, larger values of n yield better prediction accuracy, with a significant reduction in misprediction rates as mmm increases. However, the improvements taper off for values of m greater than 10, where the misprediction rate stabilizes.
- For **jpeg**, the misprediction rates are consistently lower across all configurations, with diminishing improvements as mmm increases, especially for larger values of n.

Similarities:

- Both benchmarks show a decrease in misprediction rates with increasing m across all n values.
- Both experience diminishing returns in prediction accuracy as m approaches 12.

Differences:

- The **gcc** benchmark shows more variability and a sharper decline in misprediction rates, particularly for smaller values of n.
- The **jpeg** benchmark has more stable misprediction rates and a generally lower misprediction rate overall compared to gcc.

Proposed Architecture:

For the **gcc_trace.txt** benchmark, the misprediction rate drops significantly as mmm increases. For example, when n=2n = 2n=2, the misprediction rate decreases from **28.98%** at m=7 to **12.2%** at m = 12. However, after m = 10, the decrease becomes less notable. Similarly, increasing n offers diminishing returns beyond n=4, with the misprediction rate stabilizing around **12.46%** at m=12 for n=6. Therefore, the optimal configuration for **gcc_trace.txt** is m=11 and n=4, providing a good balance between performance and storage, as further increases in m and n offer minimal improvement.

For the **jpeg_trace.txt** benchmark, the misprediction rate is less affected by changes in m and n. Starting at **8.08%** for n=2, it decreases slightly to **7.44%** at m=12. Beyond m=10, improvements are minimal, and increasing n has little impact on the misprediction rate. Thus, the best configuration for **jpeg_trace.txt** is m=10 and n=2, which optimizes performance without excessive storage or power overhead.

Both configurations fit comfortably within the storage limit. For **gcc_trace.txt**, m=11 requires **4,096 bits** for the predictor table, while **jpeg_trace.txt** requires **2,048 bits** at m=10. These values are well within the available **16kB (131,072 bits)** of storage.

In summary, the optimal configuration for **gcc_trace.txt** is m=11 and n=4, while for **jpeg_trace.txt**, it is m=10 and n=2. Both designs effectively minimize misprediction rates while staying within the 16kB storage constraint.