CS - 6304 Project 1

To simulate different benchmarks using gem5 and observe the changes in branch prediction performance of various branch predictors with change in various parameters.

Project Performed by -

Vedang Kapse - vak190002

Dhruv Mittal - dxm220015

Branch Prediction and its importance

- In a pipelined architecture multiple instructions are fetched during execution.

- While fetching instructions at a branch it is important to know if the branch will be taken or not.

- If a wrong instruction is fetched, all other instruction in the pipeline need to be flushed and execution needs to be started again.

- This adds an extreme amount of delay in pipeline execution.

- This is resolved using Branch Prediction, a technique which predicts if a branch will be taken or not, helping improvement in performance.

What is gem5?

- gem5 is an opensource software developed by combining the Michigan m5 and Wisconsin's GEMS simulator.

- gem5 is used for simulating various computer as well as processor architectures.

- gem5 is written in c++ and python which makes it easier to change the code and add custom functionalities.

Types of Predictors in gem5

There are three main types of branch predictor supported in gem5:

- Local Branch Predictor - Takes advantage of temporal locality of a branch to predict its outcome.

- BiMode Branch Predictor - Uses a n-bit state machine to predict the outcome of a branch.

- Tournament Branch Predictor - Takes advantage of spatial as well as temporal locality of a branch to predict its outcome.

Installing gem5 and dependencies

- Installing gem is a straightforward and easy task.

- One can download gem5 from its git hub repository:
  - Git clone https://gem5.googlesource.com/public/gem5

- Scons is a software compilation/build tool like Make utility which helps in faster and easier compilation.

- gem5 can be built with the help of scons.
  - Scons build/X86/gem5.opt

- gem5 build options:
  - opt – Optimized with debug symbols
  - debug
  - fast – build with all optimizations and no debug symbols
  - perf – for performance meaurement

gem5 code update for Branch Predictor

- Branch Predictor needs to be specified in - gem5/src/cpu/simple/BaseSimpleCPU.py with following line:
  - branchPred = Param.BranchPredictor(<BranchPredictor Type>, "Branch Predictor")
- Parameters for the Branch Predictor are specified in: gem5/src/cpu/BranchPredictor.py in respective predictor class. E.g.
  - localPredictorSize = Param.Unsigned(2048, "Size of local predictor")
  - globalPredictorSize = Param.Unsigned(2048, "Size of global predictor")

gem5 code change for custom stats addition

We add 2 custom stats to observe our simulations.

1. BranchMispredPct – Number of Branches mispredicted

   Declaration -
   - File changed - gem5/src/simple/exec_context.hh
   - Stats::Formula BranchMispredPct;

   Definition -
   - File Changed - gem5/src/simple/base.cc
   - t_info.BranchMispredPct = (t_info.numBranchMispred / t_info.numBranches) * 100;
   - t_info.BranchMispredPct.name(thread_str + ".BranchMispredPct") .desc("Number of branch mispredictions percentage");

2. BTBMissPct – Percentage of BTB misses

Declaration -
- File – gem5/src/cpu/pred/bpred_unit.hh
- Stats::Formula BTBMissPct;

Definition -
- File - gem5/src/cpu/pred/bpred_unit.cc
- BTBMissPct.name(name() + ".BTBMissPct")
  .desc("BTB Miss Percentage")
  .precision(6);
- BTBMissPct = (1 - (BTBHits / BTBLookups)) * 100;

Running simulation on benchmarks

- Each benchmark has its own binary and data.
- We use respective shell scripts for running the simulation on the benchmark.
- Flags used -
  - cpu-type = timing -> specifies the cpu.
  - -I = 5M -> specifies the max number of instructions to be run on the benchmark.
- Output of each simulation is stored in respective m5out dir:
  - Config.ini -> Contains configurations used for simulation e.g., Branch Predictor type, No of BTB Entries, Predictor Size etc.
  - Stats.txt -> Contains all the stats of the simulation.

Automating the simulations

- We wrote a script in python to automate the simulation of different Branch predictors, with different parameters.

- The script is attached in with the project submission.

- Steps performed by the script :-
    1. Replace BranchSimpleCPU.py with the branch predictor to be used.
    2. Replace BranchPredictor.py with required parameters for the predictor.
    3. Remove old gem5 build
    4. Compile gem5.
    5. Run simulation on the benchmarks and take backup of respective results.
    6. Repeat for all given simlation in the script.
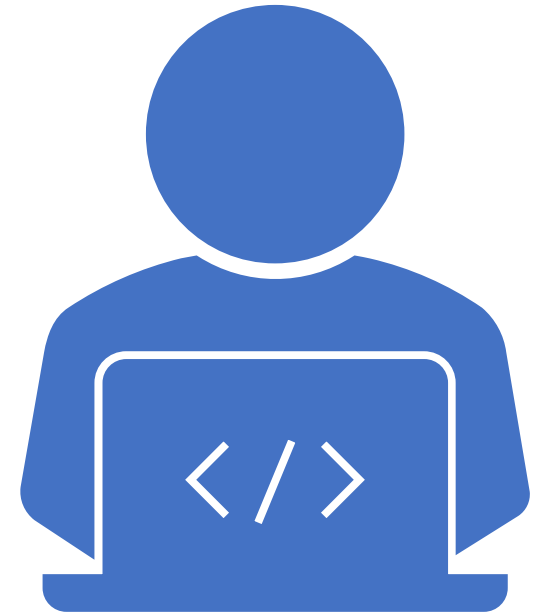
Observations on various simulations run on both benchmarks using different predictors.

Local Predictor Observations

BTBMissPct Plot for 401.bzip2

If we keep **BTBEntries** constant BTBMissPct decreases as Local predictor size increases

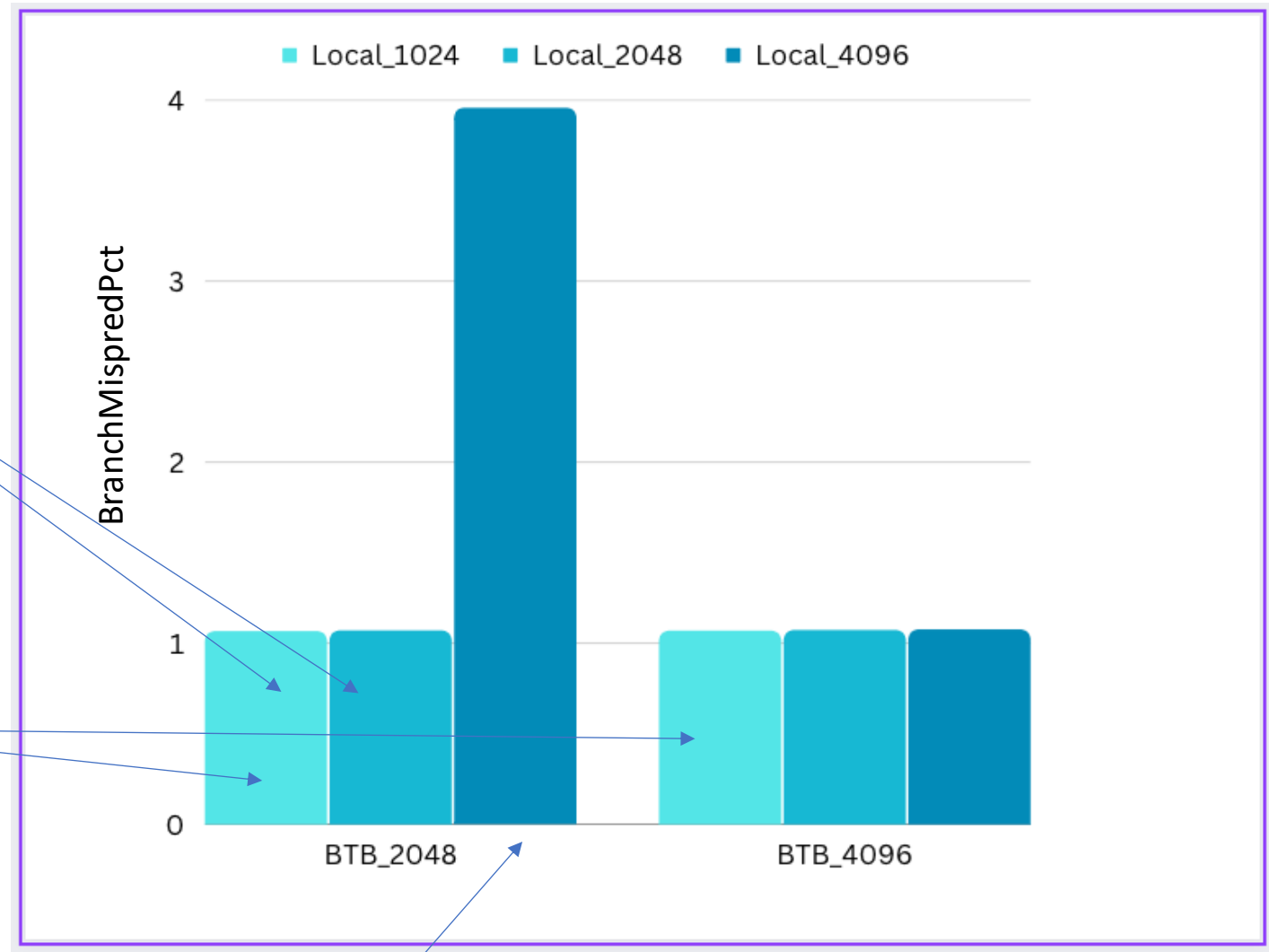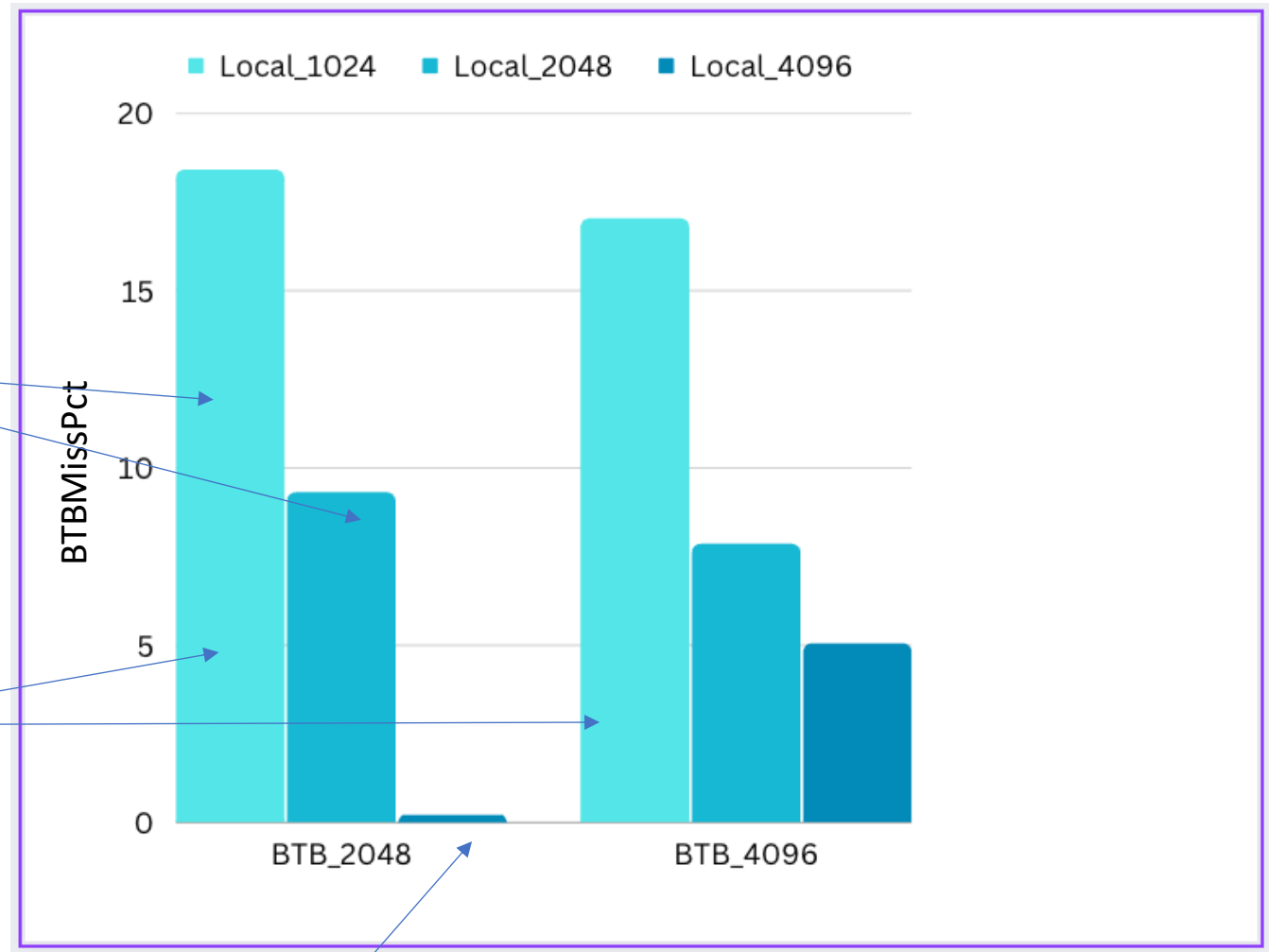If we keep **LocalPred Size** constant BTBMissPct remains constant as BTBEntries increases
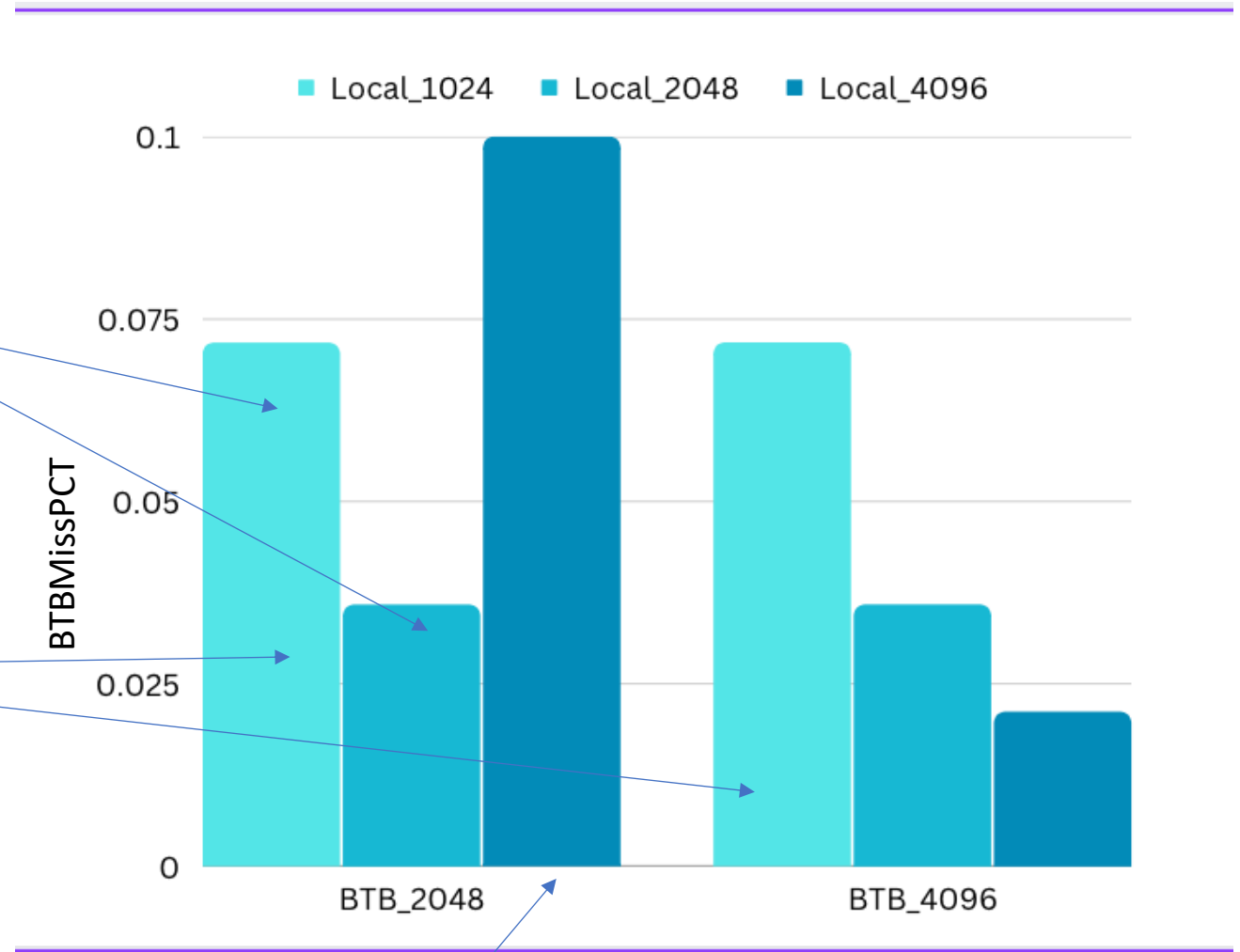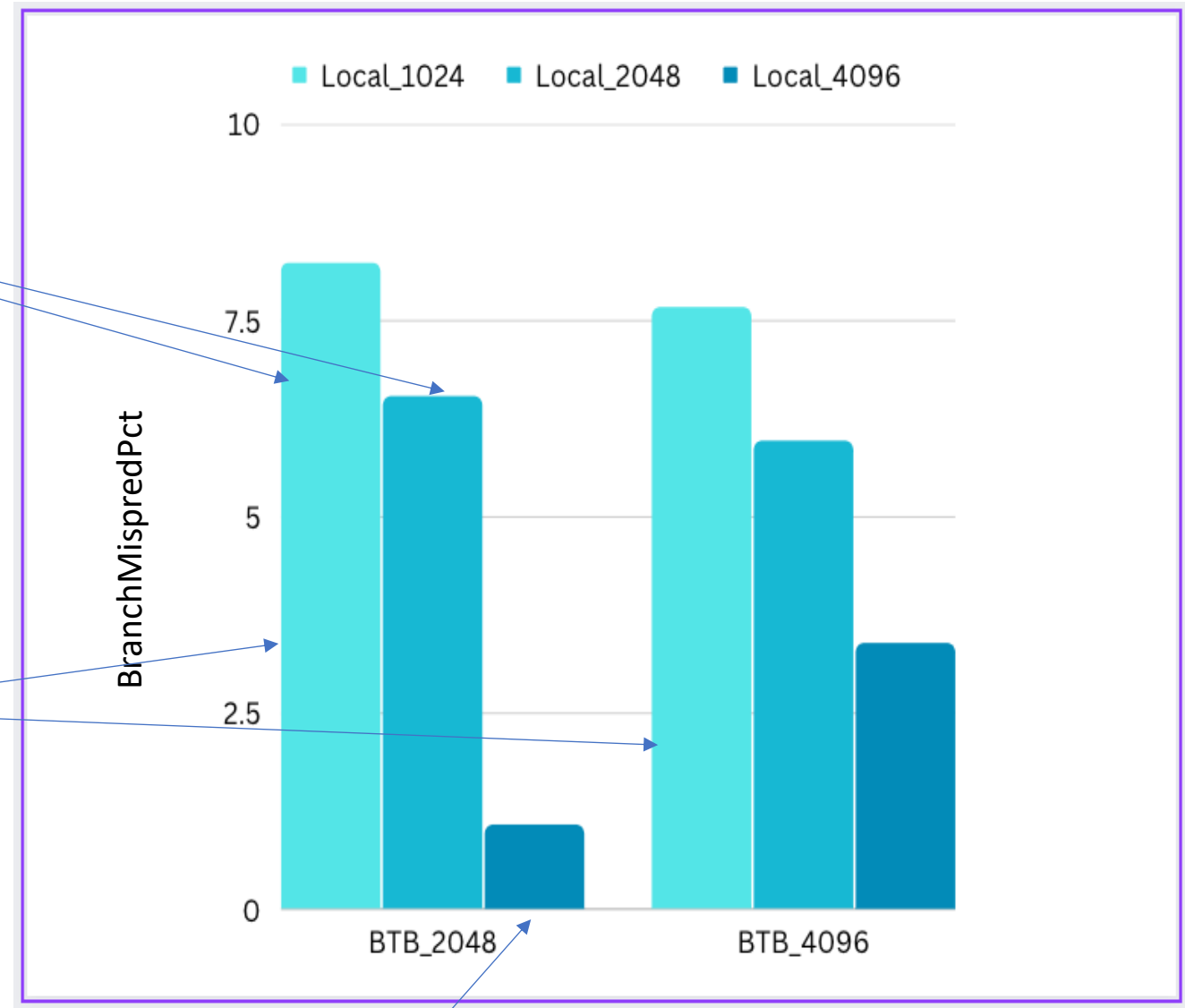

Exception

BranchMispredPct Plot for 401.bzip2

If we keep **BTBEntries** constant BranchMispredPct increases as Local predictor size increases

If we keep **LocalPred size** constant BranchMispredPct remains constant as BTBEntries increases except for LocalPred size 4096 where it decreases from BTB_2048 to BTB_4096

Exception

BTBMissPct Plot
for 429.bzip2

If we keep **BTBEntries** constant BTBMissPct decreases as Local predictor size increases except for Local_4096

If we keep **LocalPred size** constant BTBMissPct decreases as BTBEntries increase except for Local_4096

Exception

BranchMispredPct Plot for 429.bzip2

If we keep **BTBEntries** constant BranchMispredPct decreases as Local predictor size increases

If we keep **LocalPredSize** constant BranchMispredPct decreases as Local predictor size increases except for Local_4096

Exception

BiMode Predictor Observations

BTBMissPct
Plot for 401.bzip2

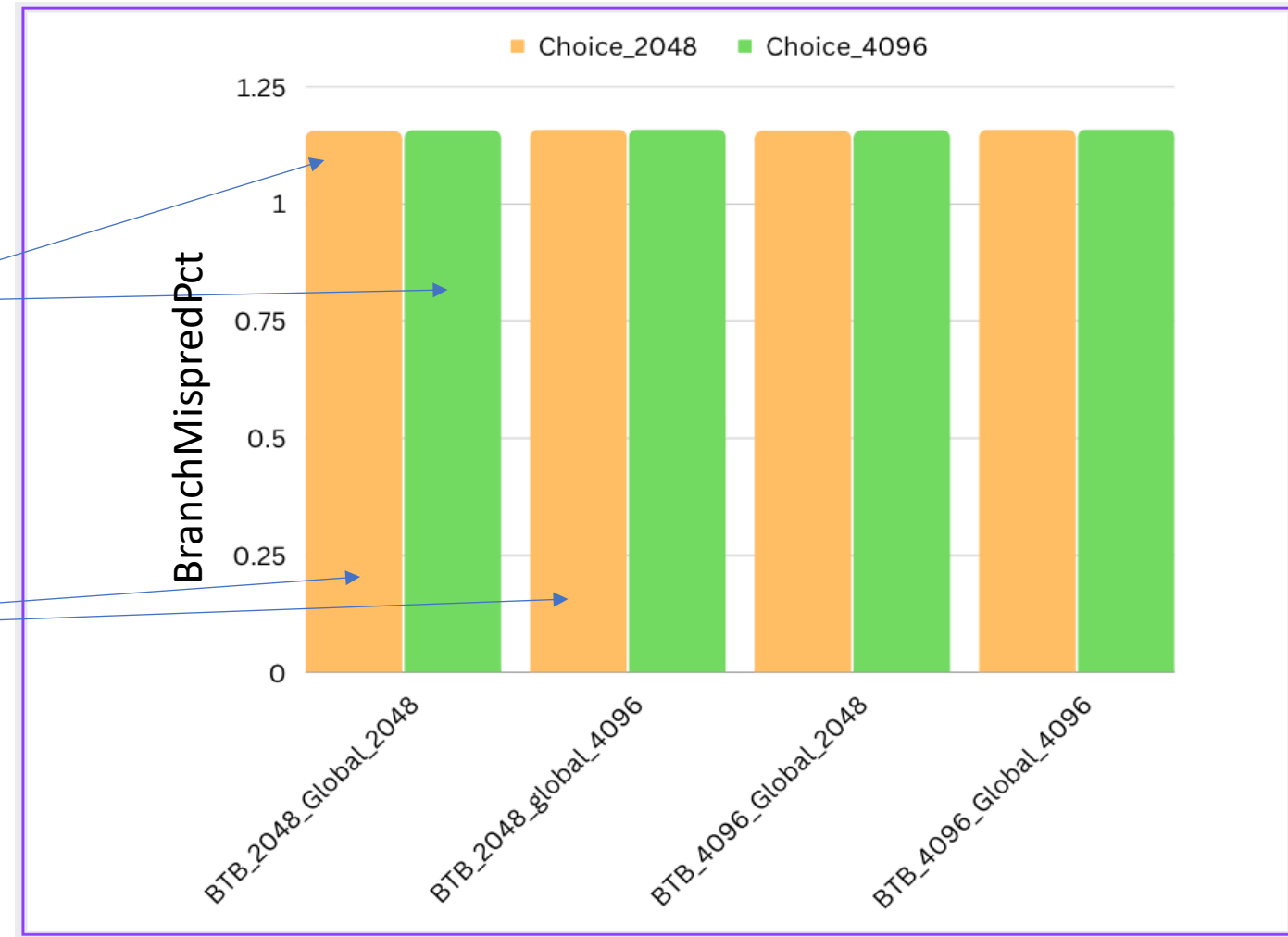If we keep **BTBEntries** and **GlobalPred** size constant BTBMissPct increases as Choice size increases

If we keep **Choice size** and **GlobalPred size** constant, BTBMissPct remains same for all BTBEntries
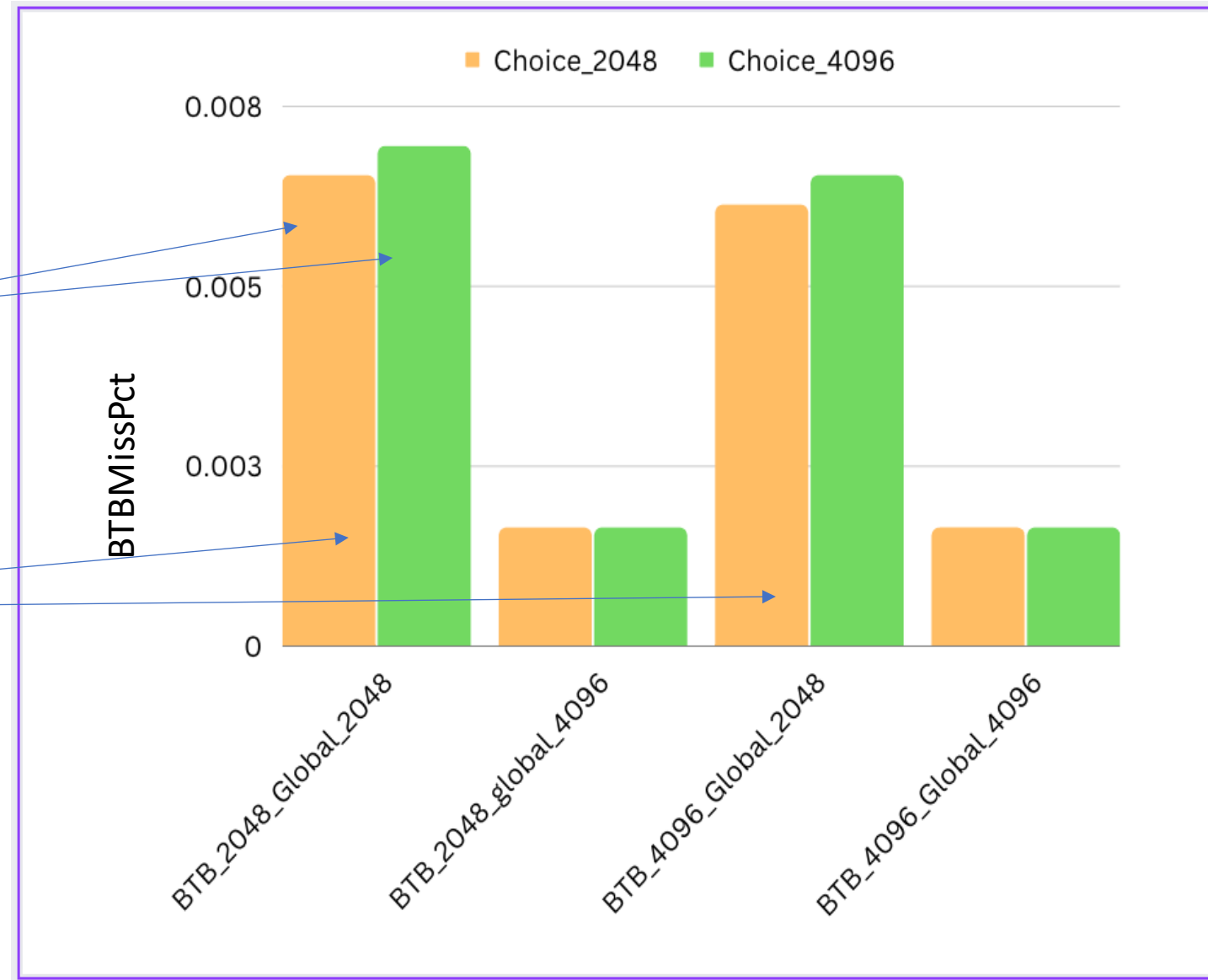
BranchMispredPct
Plot for 401.bzip2

If we keep **BTBEntries** and **GlobalPred** size constant BranchMispredPct slightly increases as Choice size increases
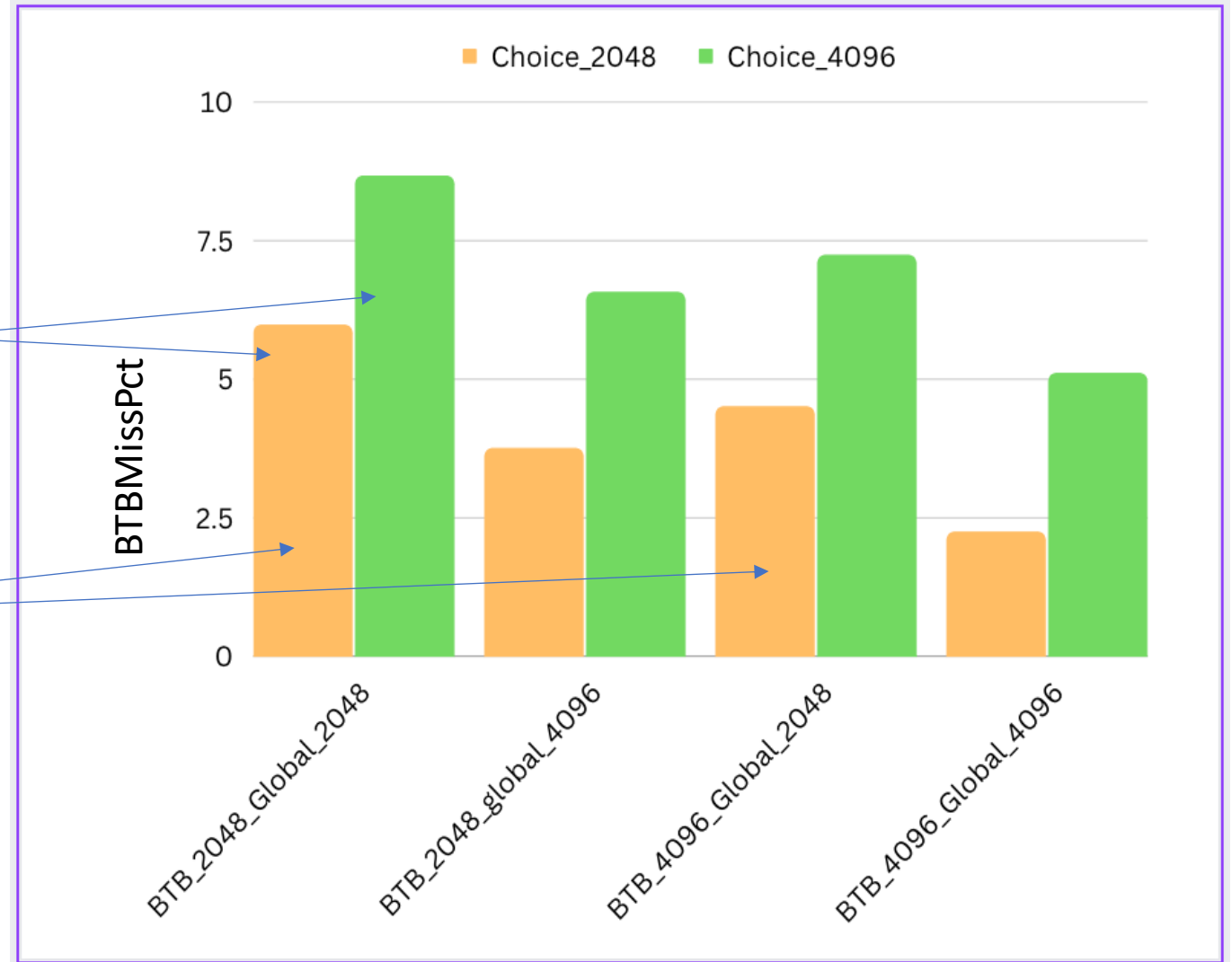
If we keep **Choice size** and **GlobalPred size** constant BranchMispredPct slightly increases as BTBEntries increases except when Global size is 4096 where it remains constant

BTBMissPct Plot for 429.mcf

If we keep **BTBEntries** and **GlobalPred** size constant BTBMissPct, increases as Choice size increases
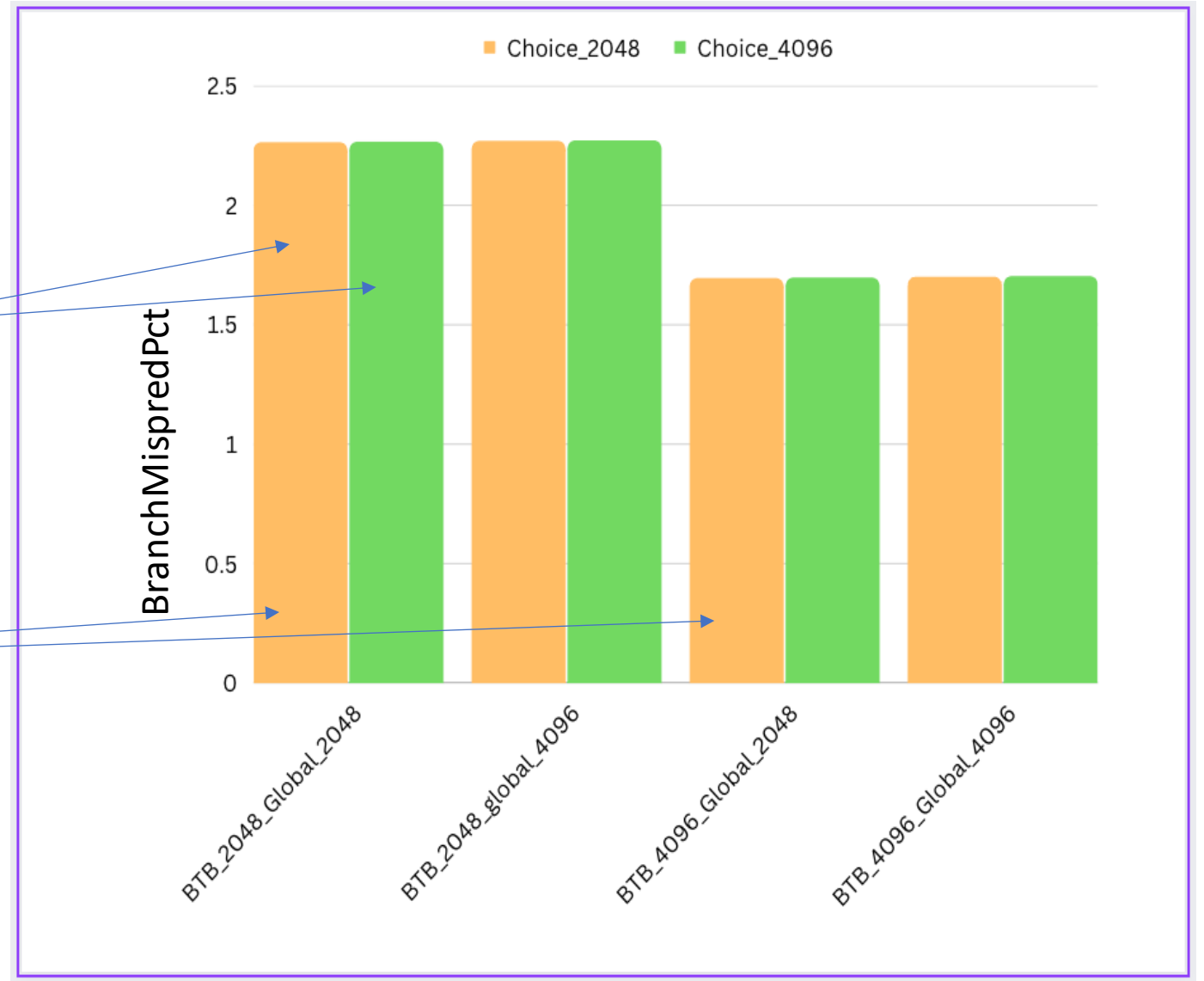
If we keep **Choice size** and **GlobalPred size** constant, BTBMissPct decreases as BTBEntries increases

BranchMispredPct Plot for 429.mcf

If we keep **BTBEntries** and **GlobalPred** size constant BranchMispredPct slightly increases as Choice size increases

If we keep **Choicesize** and **GlobalPred size** constant BranchMispredPct slightly increases as BTBEntries decreases
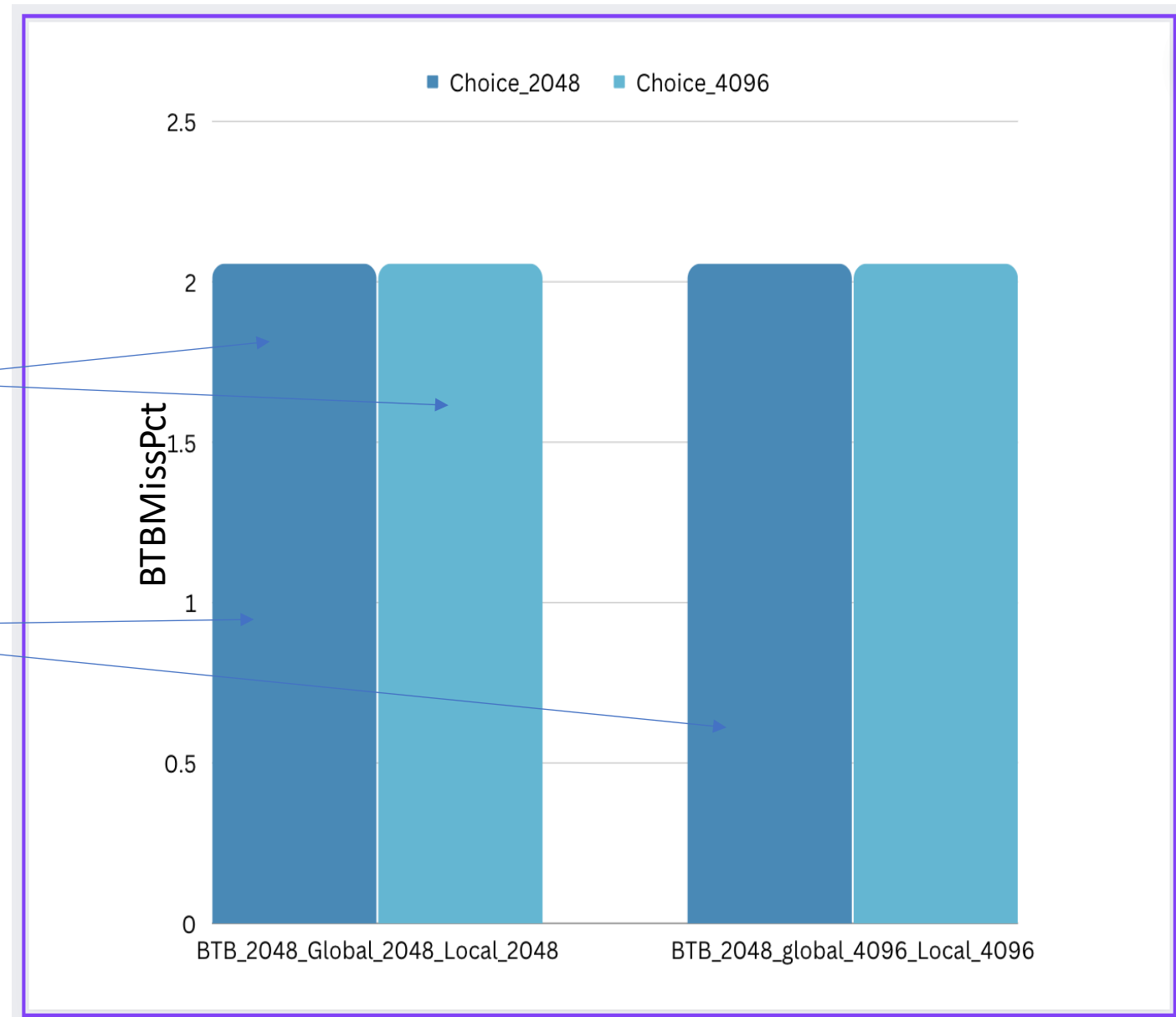
Tournament Predictor Observations

BTBMissPct Plot for 401.mcf

If we keep **BTBEntries, GlobalPred** and **LocalPred** size constant, BTBMissPct increases slightly if we increase the Choice size.
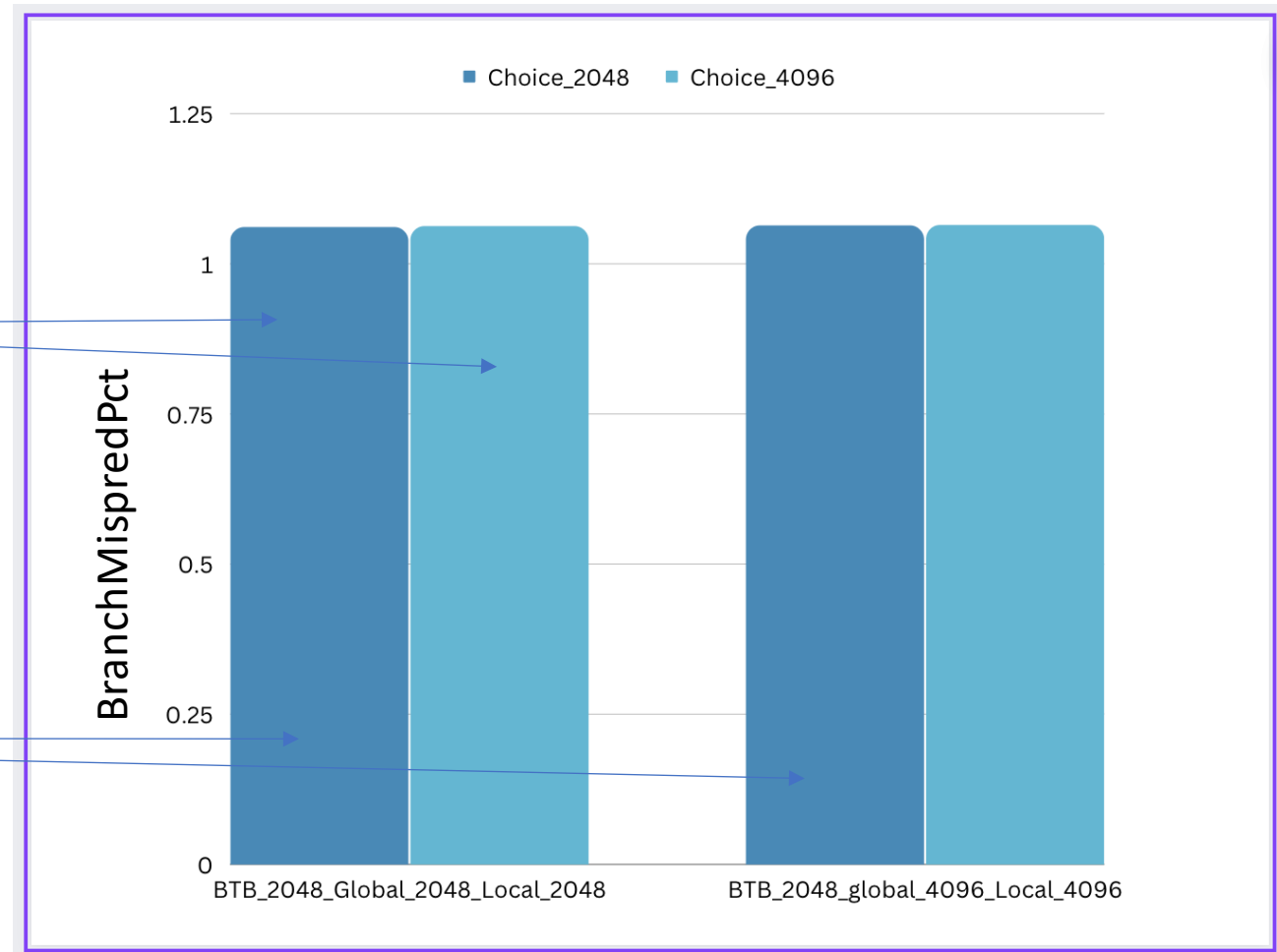
If we keep **BTBEntries** and **Choice size** constant, if we increase in Global and Local Pred Size together, BTBMissPct slightly increases.

BranchMispredPct plot for 401.bzip2

If we keep **BTBEntries, GlobalPred** and **LocalPred** size constant, BranchMispredPct increases slightly if we increase the Choice size.
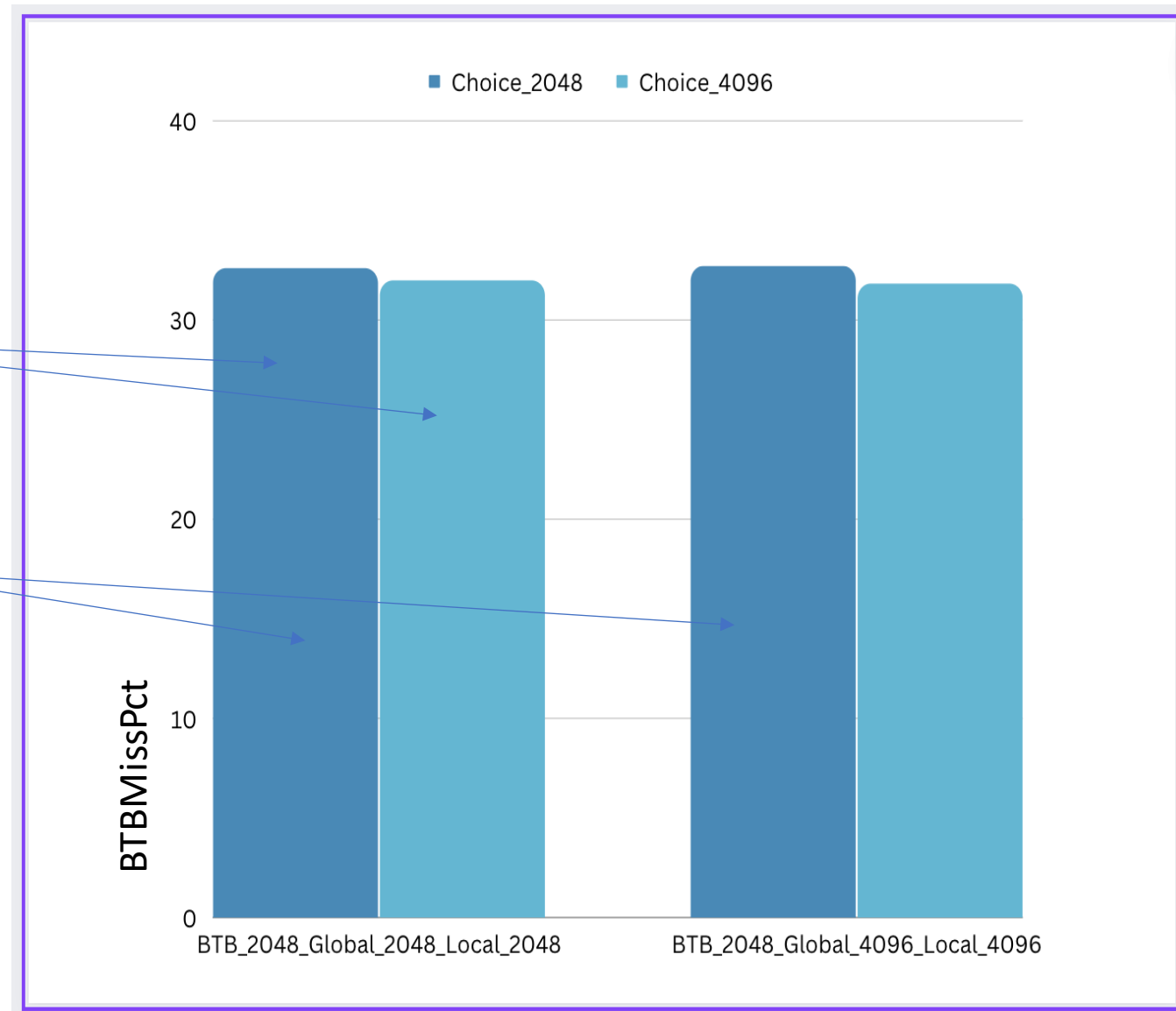
If we keep **BTBEntries** and **Choice size** constant, increase in Global and Local Pred Size together, BranchMispredPct slightly increases.

BTBMissPct plot for 429.mcf

If we keep **BTBEntries, GlobalPred** and **LocalPred** size constant, BTBMissPct decreases slightly if we increase the Choice size.
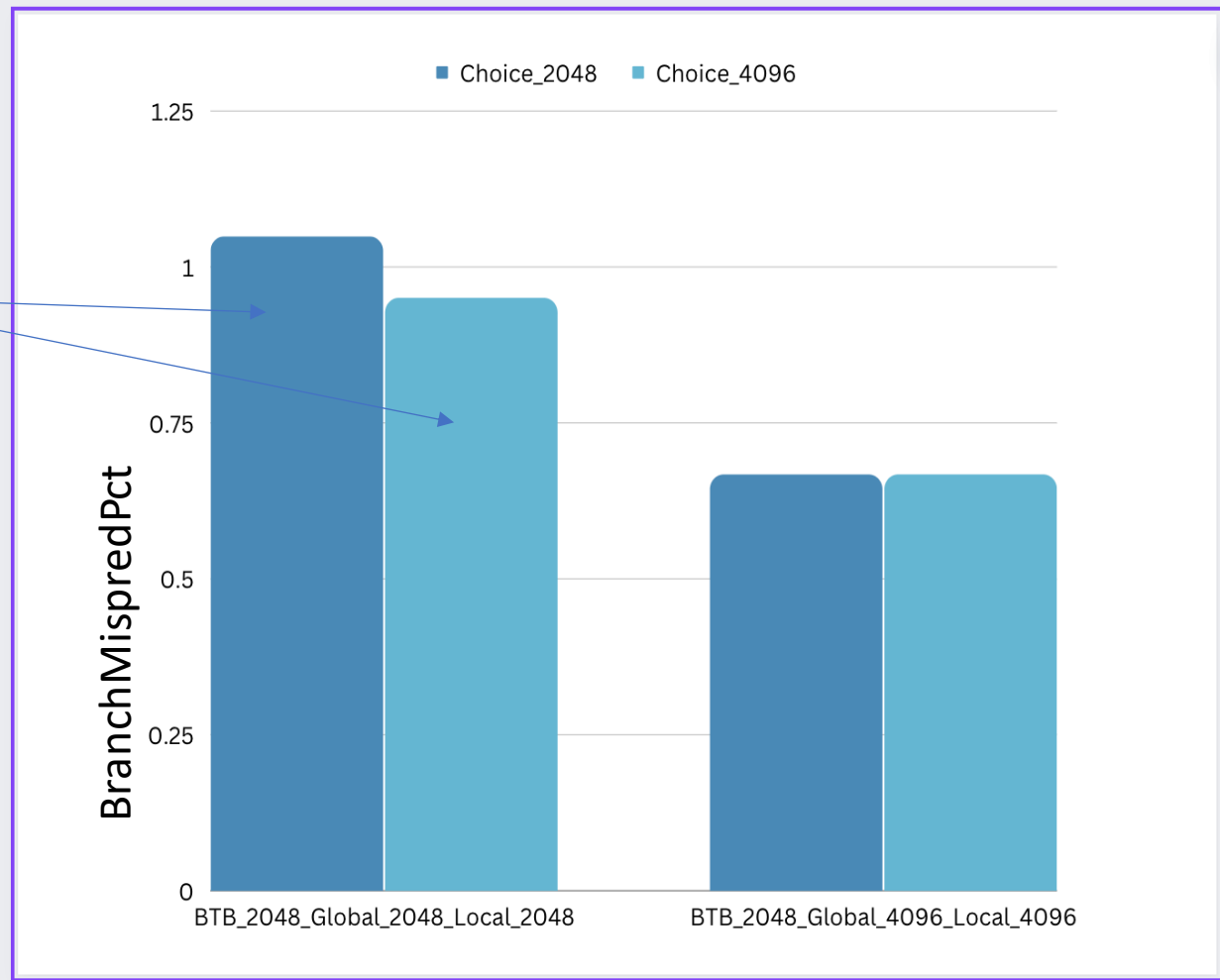
If we keep **BTBEntries** and **Choice size** constant, if we increase in Global and Local Pred Size together, BTBMissPct slightly decreases.

BranchMispredPct plot for 429.mcf

If we keep **BTBEntries, GlobalPred** and **LocalPred** size constant, BranchMispredPct increases slightly if we increase the Choice size.
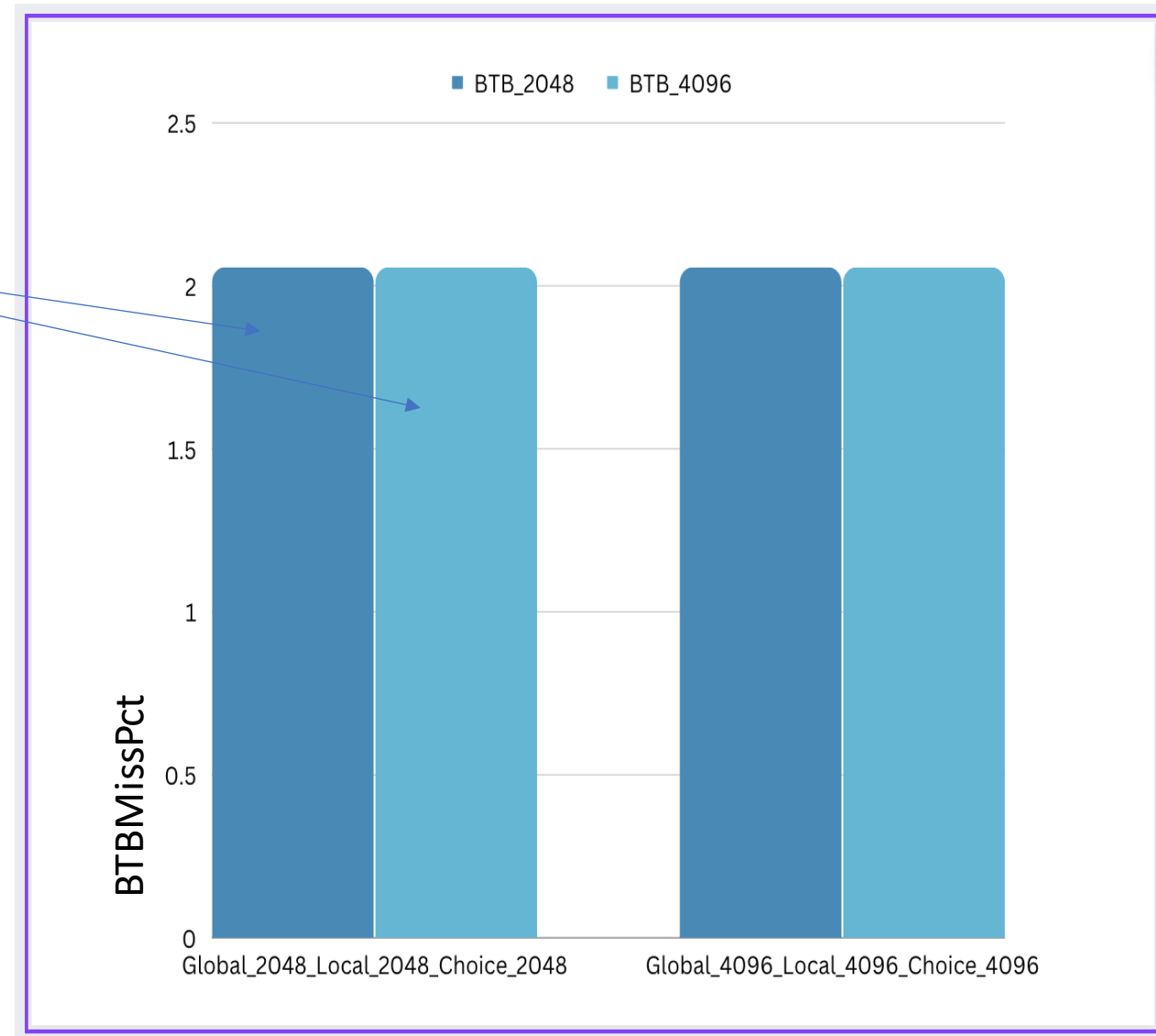
If we keep **BTBEntries, GlobalPred** and **LocalPred** size constant, BranchMispredPct increases slightly if we increase the Choice size.

BTBMissPct plot for 401.bzip2

If we keep **Choice, GlobalPred** and **LocalPred** size constant, BTBMissPct does not change much with change in BTB Entries.
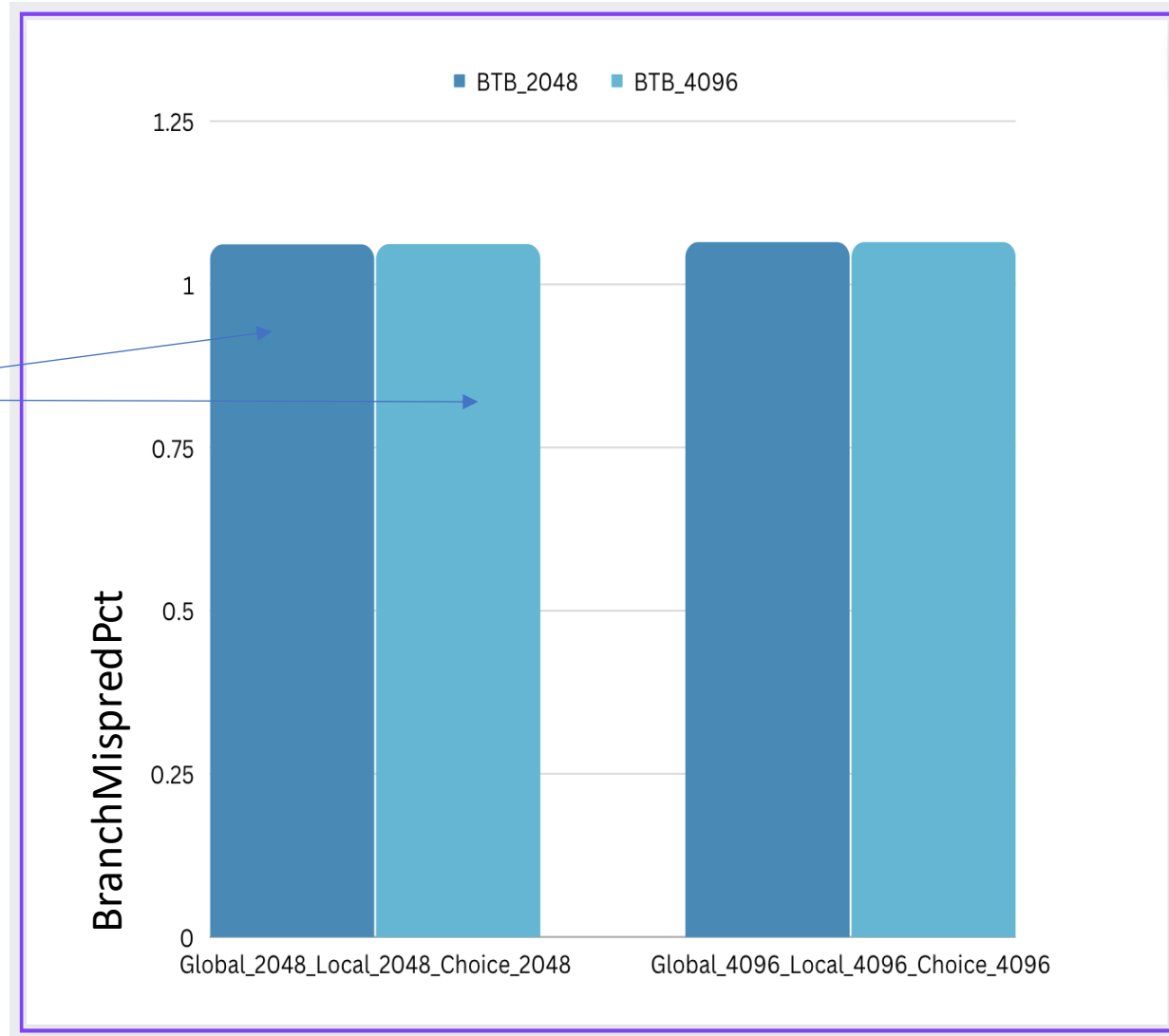
i.e. change in BTBEntries does not affect the BTBMissPct much.

BranchMispredPct plot for 401.bzip2

If we keep **Choice, GlobalPred** and **LocalPred** size constant, BranchMispredpct does not change much with change in BTB Entries.
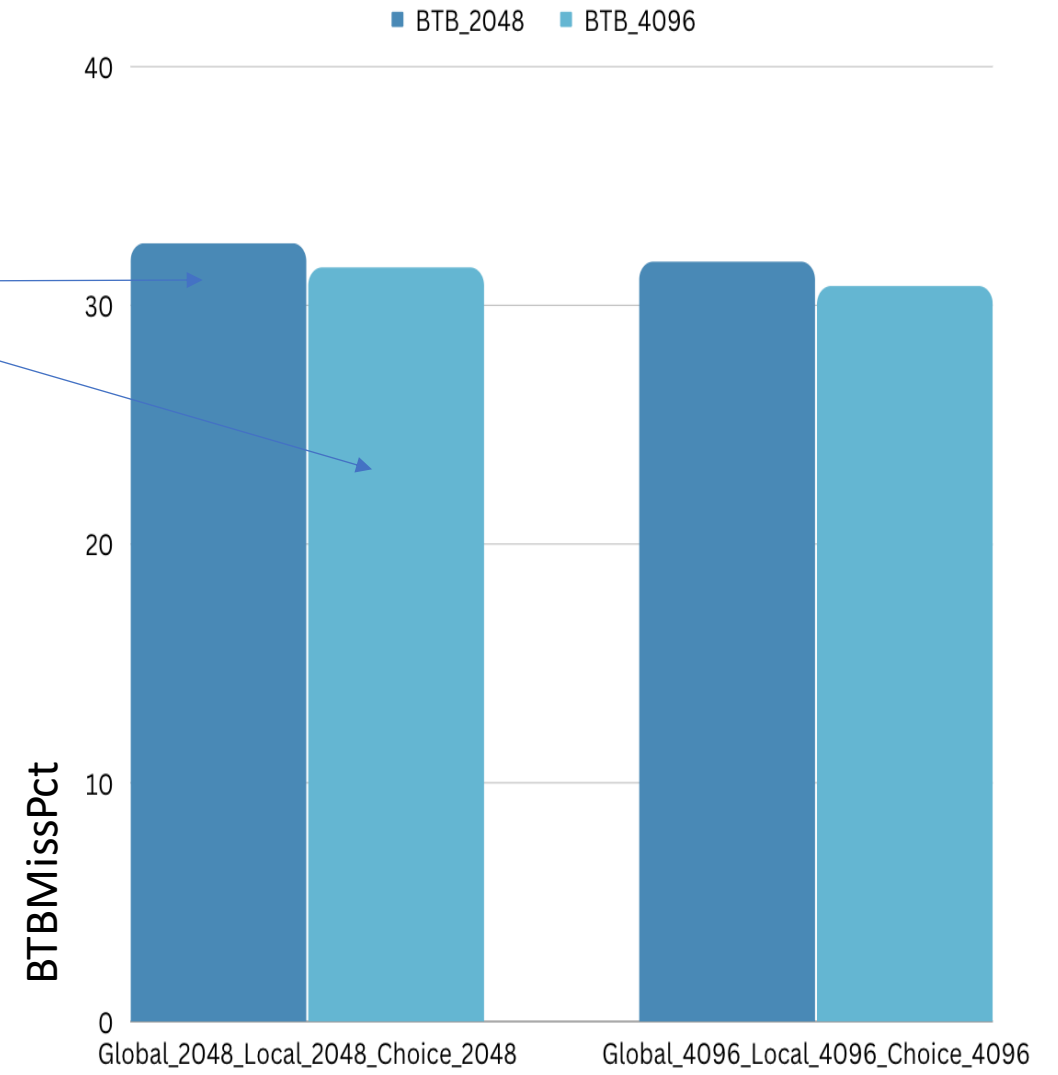
i.e. change in BTBEntries does not affect the BranchMispredPct much.

BTBMissPct plot for 429.mcf

If we keep **Choice, GlobalPred** and **LocalPred size** constant, BTBMissPct decreases with increase in BTB Entries.

It is only reasonable since increasing the BTB size should means we can store more addresses in the buffer thus reducing the BTB misses.
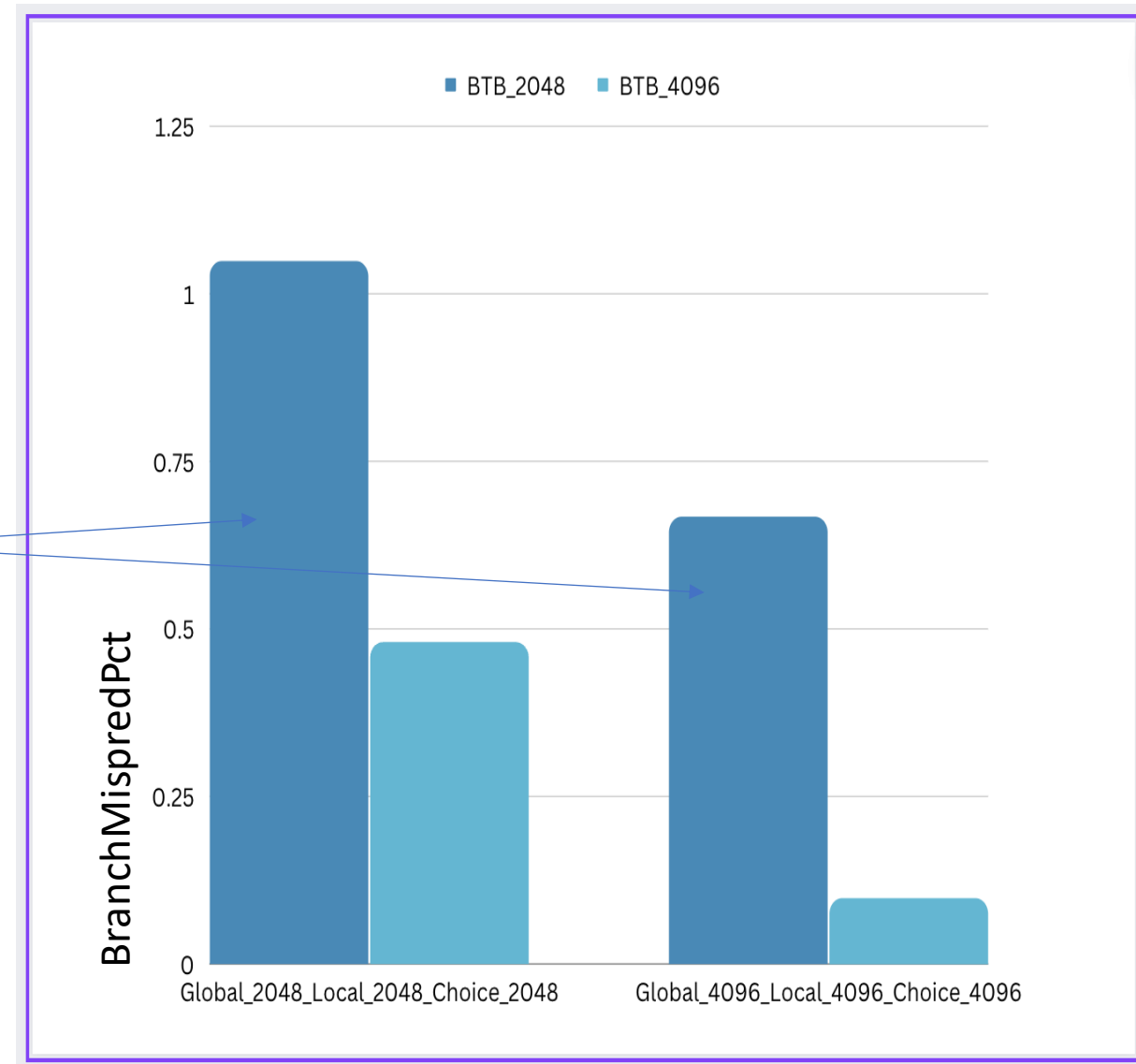
BranchMispredPct plot for 429.mcf

If we keep **Choice, GlobalPred** and **LocalPred size** constant, BranchMispredPct decreases with increase in BTB Entries.

We also observe that with increase in size of all 3 predictors there is tremendous decrease in BranchMispredPct.

This is because with increase in predictor size the accuracy of predictor increases.

Final Observations for 401.bzip2

- Increasing BTBEntries does not have a major effect the BTBMissPct, probably because there are not as many branches to enter in the branch target buffer or not many branches need to be fetched again.

- Tournament Predictor gives the best accuracy i.e. least BranchMispredictions as it makes use both temporal and spatial locality of a branch.

- However, we also observe that the MispredPct changes from 1.067 to 1.060 from Local to Tournament, suggesting there might not be as many branches in the benchmark.

- BiModal has the worst accuracy for the given benchmark.

Final Observations for 429.mcf

- Increasing the number of BTBEntries decreases the BTBMissPct by around 1-1.5% for all predictors, since with increase in BTB size we can store targets of more branches in the table.

- This decrease is more than that observed in 401.bzip 2, probably because 429.mcf has more branches.

- Tournament predictor performs extremely well on the benchmark since it makes use of temporal as well as spatial locality.

- BiModal predictor performs the worst.

- The BranchMispredPct changes from around 1.7 to 1.07 to 0.09 as we change from BiModal to Local to Tournament predictor respectively.

- This increase in accuracy is major and more than that compared to 401.bzip2 benchmark, suggesting that 429.mcf has a lot of branches.