Step 1: git setup, created script for future assignments repo creation, fixed the password required thing every call without SSH with help of aditya  
  
step2:  
basically connected gem5 repo so dont have to copy paste huge file set, we can use it from already available version, next started building X86

able to run “Hello world” using  
chirag@chirag:~/HPC-Projects/gem5$ ./build/X86/gem5.opt configs/deprecated/example/se.py \

-c tests/test-progs/hello/bin/x86/linux/hello \

--cpu-type=O3CPU --bp-type=TournamentBP \

--caches –l2cache

gem5 Simulator System. https://www.gem5.org

gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.0.0.1

gem5 compiled Sep 6 2025 02:38:56

gem5 started Sep 6 2025 03:04:05

gem5 executing on chirag, pid 20811

command line: ./build/X86/gem5.opt configs/deprecated/example/se.py -c tests/test-progs/hello/bin/x86/linux/hello --cpu-type=O3CPU --bp-type=TournamentBP --caches --l2cache

warn: The se.py script is deprecated. It will be removed in future releases of gem5.

Global frequency set at 1000000000000 ticks per second

src/mem/dram\_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)

src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.

system.remote\_gdb: Listening for connections on port 7000

\*\*\*\* REAL SIMULATION \*\*\*\*

Hello world!

Exiting @ tick 20592000 because exiting with last active thread context

[chirag@chirag](mailto:chirag@chirag):~/HPC-Projects/gem5$  
  
  
  
If it works, you’ll see Hello world! in the terminal and stats written to m5out/stats.txt.

Open the stats:

less m5out/stats.txt

### What happens under the hood

* --caches → turns on **L1 instruction cache** and **L1 data cache**.
* --l2cache → adds an **L2 cache**, but it assumes that L1 caches already exist.
* If you only pass --l2cache without --caches, the CPU has **no L1 caches at all**, and gem5 doesn’t know how to connect O3CPU → memory → L2. That’s why it crashes.

So:

* **Valid:** --caches (O3 + L1)
* **Valid:** --caches --l2cache (O3 + L1 + L2)
* **Invalid:** --l2cache only (missing L1 caches)

Why does O3CPU require caches? Why doesn’t --l2cache alone work?

You can say:

* The O3CPU in gem5 models realistic out-of-order execution and depends on instruction/data caches for timing.
* L1 caches are mandatory because they connect the CPU to the memory hierarchy.
* L2 cache is optional but only makes sense if L1s exist — otherwise, the CPU would have no direct interface.

That’s a **good conceptual point** to show you understand gem5’s design.

Test if it works :

## Test gem5 manually one more time

Run hello-world with O3CPU + TournamentBP + caches:

cd ~/HPC-Projects/gem5

./build/X86/gem5.opt configs/deprecated/example/se.py \

-c tests/test-progs/hello/bin/x86/linux/hello \

--cpu-type=O3CPU --bp-type=TournamentBP \

--caches --l2cache

Expected:

* Terminal prints Hello world!
* m5out/stats.txt gets generated

Quick check:

grep sim\_ipc m5out/stats.txt

grep branchPred m5out/stats.txt | head -20

If you see IPC and branch prediction numbers → ✅ build + run are solid.

Step 3: baseline is ready tell Chirag to do  
git clone [git@github.com](mailto:git@github.com):Kathpaliachirag/A2-gem5-HPC.git  
  
  
Perfect 👍 let’s move on to **Step 3 — writing your own wrapper script in your repo**.  
This step will make running experiments **easy + reproducible + collaborative**.

## 🎯 Goal of Step 3

* Avoid typing long gem5.opt ... commands manually.
* Save each experiment’s output (stats.txt, config.ini, etc.) into your repo under results/raw/.
* Make it easy to extend later (different predictors, benchmarks).

## ****1) Create a run script****

Go to your assignment repo:

cd ~/HPC-Projects/gem5-branch-prediction/scripts

Create a script file:

nano run\_experiment.sh

Paste this:

#!/bin/bash

# run\_experiment.sh

# Wrapper script to run gem5 with O3CPU + branch predictor

# Saves results into ../results/raw/

# -------------------

# Arguments

# -------------------

BP=$1 # branch predictor (e.g., TournamentBP, LocalBP, BiModeBP)

OUTDIR="../results/raw/${BP}"

# -------------------

# Paths (adjust if your gem5 folder is elsewhere)

# -------------------

GEM5=~/HPC-Projects/gem5/build/X86/gem5.opt

SE=~/HPC-Projects/gem5/configs/deprecated/example/se.py

PROG=~/HPC-Projects/gem5/tests/test-progs/hello/bin/x86/linux/hello

# -------------------

# Run

# -------------------

mkdir -p $OUTDIR

$GEM5 \

--outdir=$OUTDIR \

$SE \

-c $PROG \

--cpu-type=O3CPU \

--bp-type=$BP \

--caches --l2cache

Save (Ctrl+O, Enter, Ctrl+X).

Make it executable:

chmod +x run\_experiment.sh

## ****2) Try running it****

Run with TournamentBP:

./run\_experiment.sh TournamentBP

Check outputs:

ls ../results/raw/TournamentBP

You should see:

* stats.txt
* config.ini
* config.json
* simout, simerr, etc.

## ****3) Read the stats****

Check IPC and mispredictions:

grep sim\_ipc ../results/raw/TournamentBP/stats.txt

grep branchPred ../results/raw/TournamentBP/stats.txt | head -20

## ****4) Commit your script****

Go back to repo root and commit:

cd ..

git add scripts/run\_experiment.sh results/raw/.gitkeep

git commit -m "feat: add run\_experiment.sh for O3CPU + BP experiments"

git push

✅ Now you have:

* A reusable wrapper script (scripts/run\_experiment.sh)
* Results for each BP neatly separated in results/raw/

👉 Next (Step 4), we’ll extend the script to:

1. Run **multiple branch predictors automatically** (LocalBP, TournamentBP, BiModeBP, etc.).
2. Extract key stats (IPC, mispred rate) into a CSV inside results/summary/ → ready for plotting/report.

Would you like me to show you **Step 4 (multi-BP runs + CSV extraction)** now?

Perfect 🚀 let’s move on to **Step 4 — batch runs + extracting stats into CSV**.  
This will give you structured data (IPC + mispred rate per predictor) for analysis and plots.

# 🎯 Step 4 Goals

1. Extend your run\_experiment.sh so it can run **multiple branch predictors in one go**.
2. Write a **Python parser** to extract IPC and mispred rate from each stats.txt.
3. Store results in results/summary/results.csv.

## ****1) Extend Bash script for multiple BPs****

Edit your script:

cd ~/HPC-Projects/A2-gem5-HPC/scripts

nano run\_experiment.sh

Replace with this:

#!/bin/bash

# run\_experiment.sh

# Run gem5 O3CPU with multiple branch predictors

# Saves results in ../results/raw/<BP>/

# -------------------

# Config

# -------------------

GEM5=~/HPC-Projects/gem5/build/X86/gem5.opt

SE=~/HPC-Projects/gem5/configs/deprecated/example/se.py

PROG=~/HPC-Projects/gem5/tests/test-progs/hello/bin/x86/linux/hello

# Branch predictors to test

BPS=("TournamentBP" "LocalBP" "BiModeBP" "LTage")

# -------------------

# Run each predictor

# -------------------

for BP in "${BPS[@]}"; do

OUTDIR="../results/raw/${BP}"

echo ">>> Running $BP"

mkdir -p $OUTDIR

$GEM5 \

--outdir=$OUTDIR \

$SE \

-c $PROG \

--cpu-type=O3CPU \

--bp-type=$BP \

--caches --l2cache

done

Save + exit.  
Make sure it’s executable (if not already):

chmod +x run\_experiment.sh

Now run all predictors:

./run\_experiment.sh

This will create:

results/raw/TournamentBP/stats.txt

results/raw/LocalBP/stats.txt

results/raw/BiModeBP/stats.txt

results/raw/LTage/stats.txt

## ****2) Parse stats into CSV****

Now let’s add a parser script in Python:

cd ~/HPC-Projects/A2-gem5-HPC/scripts

nano parse\_stats.py

Paste this:

#!/usr/bin/env python3

import os

import re

import csv

RAW\_DIR = "../results/raw"

OUT\_FILE = "../results/summary/results.csv"

def extract\_stats(stats\_file):

ipc = None

lookups = 0

mispred = 0

with open(stats\_file) as f:

for line in f:

if "sim\_ipc" in line or "system.cpu.ipc" in line:

ipc = float(line.split()[1])

if "branchPred.lookups\_0::total" in line:

lookups = int(line.split()[1])

if "branchPred.squashes\_0::total" in line:

mispred = int(line.split()[1])

mispred\_rate = (mispred / lookups) if lookups > 0 else 0

return ipc, mispred\_rate

def main():

predictors = []

rows = []

for bp in os.listdir(RAW\_DIR):

stats\_file = os.path.join(RAW\_DIR, bp, "stats.txt")

if os.path.isfile(stats\_file):

ipc, mispred\_rate = extract\_stats(stats\_file)

predictors.append(bp)

rows.append((bp, ipc, mispred\_rate))

with open(OUT\_FILE, "w", newline="") as f:

writer = csv.writer(f)

writer.writerow(["BranchPredictor", "IPC", "MispredRate"])

writer.writerows(rows)

print(f"✅ Results written to {OUT\_FILE}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

Save + exit.  
Make it executable:

chmod +x parse\_stats.py

Run parser:

./parse\_stats.py

Check results:

cat ../results/summary/results.csv

You should see something like:

BranchPredictor,IPC,MispredRate

TournamentBP,1.23,0.04

LocalBP,1.10,0.08

BiModeBP,1.05,0.09

LTage,1.30,0.02

## ✅ End of Step 4

Now you have:

* Automated batch runs for multiple predictors
* A CSV file summarizing IPC + misprediction rate
* A workflow that’s repeatable and sharable with your teammate

👉 Next Step 5 could be: plotting these results into graphs (bar chart IPC vs predictor, line chart mispred rate) and saving plots into results/summary/ for your report.

Do you want me to take you through **Step 5 (plotting CSV results)** next?

Perfect 🚀 — let’s go to **Step 5: plotting results**.  
At this stage, you already have results/summary/results.csv with IPC + misprediction rates for each branch predictor. Now we’ll turn that into graphs.

# 🎯 Step 5 Goals

1. Write a Python plotting script using **matplotlib**.
2. Generate bar charts for **IPC** and **Misprediction Rate**.
3. Save plots in results/summary/ so they can go straight into your report.

## ****1) Create plotting script****

Inside your scripts/ folder:

cd ~/HPC-Projects/A2-gem5-HPC/scripts

nano plot\_results.py

Paste this:

#!/usr/bin/env python3

import pandas as pd

import matplotlib.pyplot as plt

# -------------------

# Config

# -------------------

CSV\_FILE = "../results/summary/results.csv"

OUT\_DIR = "../results/summary"

# -------------------

# Load data

# -------------------

df = pd.read\_csv(CSV\_FILE)

# -------------------

# Plot IPC

# -------------------

plt.figure(figsize=(6,4))

plt.bar(df["BranchPredictor"], df["IPC"])

plt.title("IPC for Different Branch Predictors")

plt.ylabel("Instructions Per Cycle (IPC)")

plt.xlabel("Branch Predictor")

plt.tight\_layout()

plt.savefig(f"{OUT\_DIR}/ipc\_comparison.png")

plt.close()

# -------------------

# Plot Misprediction Rate

# -------------------

plt.figure(figsize=(6,4))

plt.bar(df["BranchPredictor"], df["MispredRate"])

plt.title("Misprediction Rate for Different Branch Predictors")

plt.ylabel("Misprediction Rate")

plt.xlabel("Branch Predictor")

plt.tight\_layout()

plt.savefig(f"{OUT\_DIR}/mispred\_rate\_comparison.png")

plt.close()

print("✅ Plots saved in results/summary/")

Save & exit.  
Make it executable:

chmod +x plot\_results.py

## ****2) Run the plotting script****

./plot\_results.py

Check:

ls ../results/summary/

You should see:

* results.csv
* ipc\_comparison.png
* mispred\_rate\_comparison.png

## ****3) Preview plots****

If you open them (e.g. via Windows File Explorer in your WSL project folder), you’ll see:

* **IPC bar chart** → higher bars mean better performance
* **Misprediction rate bar chart** → lower bars mean better accuracy

## ✅ End of Step 5

Now you have:

* Raw results (stats.txt)
* Summary table (results.csv)
* Visuals (ipc\_comparison.png, mispred\_rate\_comparison.png)

Your repo is now producing publishable results 💯.

how do I open images from terminal?

## 2. Open directly with an image viewer in Linux (if installed)

If you installed something like **feh** or **eog** (Eye of GNOME):

sudo apt install -y feh

feh ../results/summary/ipc\_comparison.png

But most WSL users don’t have a Linux desktop environment, so this often won’t work out-of-the-box.

👉 Next (Step 6), we can:

* Automate a **full pipeline script** (make all) that runs predictors → parses → plots in one command,  
  or
* Start preparing the **report (LaTeX/Markdown)** where you’ll include these plots.

Which one do you want to do next?

Awesome 👌 you’re moving like a pro now.  
**Step 6 = Automate the full pipeline** → one command runs everything: batch predictors → parse stats → generate plots. This is what you’ll demo in viva for reproducibility.

# 🎯 Step 6 Goals

1. Create a **master script** (run\_all.sh)
2. It should:
   * Run all branch predictors
   * Parse results into CSV
   * Generate plots
3. Save outputs neatly in results/

## ****1) Create master script****

Go to your scripts/ folder:

cd ~/HPC-Projects/A2-gem5-HPC/scripts

nano run\_all.sh

Paste this:

#!/bin/bash

# run\_all.sh

# Full pipeline: run predictors -> parse stats -> plot results

echo ">>> [1/3] Running gem5 experiments..."

./run\_experiment.sh

echo ">>> [2/3] Parsing stats into CSV..."

./parse\_stats.py

echo ">>> [3/3] Generating plots..."

./plot\_results.py

echo "✅ All steps complete. Check results in ../results/summary/"

Save + exit, then make executable:

chmod +x run\_all.sh

## ****2) Run full pipeline****

Now, instead of 3 separate steps, just:

./run\_all.sh

This will:

* Run predictors (TournamentBP, LocalBP, BiModeBP, LTage)
* Collect stats → results/summary/results.csv
* Produce plots → results/summary/ipc\_comparison.png + mispred\_rate\_comparison.png

## ****3) Verify****

Check:

ls ../results/summary/

You should see:

results.csv

ipc\_comparison.png

mispred\_rate\_comparison.png

## ✅ End of Step 6

Now your repo is **turnkey reproducible**:

* Anyone (your teammate, examiner) can run:

cd scripts

./run\_all.sh

→ and instantly get results + plots.

👉 Next (Step 7), we can:

* 📑 Start drafting the **report** (LaTeX/Markdown) with figures, or
* ⚡ Improve experiments (run on bigger workloads instead of just hello-world).

Would you like to go into **report writing** next, or **add more realistic benchmarks** to your experiments?

Ahhh got it 👍 — if your instructor explicitly listed **PARSEC** and **MiBench** tutorials, then using them will look more professional in your viva than just custom C programs.

That said, there’s a trade-off we need to manage:

* **PARSEC / MiBench** = industry-standard benchmarks, but heavy → gem5 runs can take **hours or even days** unless you run **short / reduced inputs**.
* **Custom kernels** = quick, but look less “official”.

Since you want to use **PARSEC + MiBench**, let’s do it the right way but keep workloads small enough to run in gem5 SE mode.

## 🔎 Step 7 Plan (with PARSEC & MiBench)

1. **Get the benchmark suites** inside WSL.
   * PARSEC Tutorial (small workloads available).
   * MiBench Tutorial (embedded workloads).
2. **Build a couple of programs** from each suite.
   * From PARSEC: blackscholes (compute-heavy, financial kernel).
   * From MiBench: automotive/basicmath (branch-heavy with conditionals).
3. **Integrate into your pipeline** as workloads.
   * Store compiled binaries in workloads/parsec/ and workloads/mibench/.
   * Scripts loop over both predictors **and** workloads.

## 🔧 1. Install dependencies

Inside WSL, install tools needed for building benchmarks:

sudo apt update

sudo apt install -y gcc g++ make cmake unzip

## 🔧 2. Download benchmark suites

* **PARSEC tutorial** (smaller set, easier than full PARSEC):

git clone https://github.com/bamos/parsec-benchmark.git ~/HPC-Projects/workloads/parsec

Then:

cd ~/HPC-Projects/workloads/parsec

source env.sh

Build blackscholes:

parsecmgmt -a build -p blackscholes

* **MiBench**:

git clone https://github.com/embecosm/mibench.git ~/HPC-Projects/workloads/mibench

Build basicmath:

cd ~/HPC-Projects/workloads/mibench/automotive/basicmath

make

→ produces binary basicmath\_small or basicmath\_large.

## 🔧 3. Integrate into gem5 pipeline

Your workloads directory will look like:

workloads/

parsec/blackscholes/inst/amd64-linux.gcc/bin/blackscholes

mibench/automotive/basicmath/basicmath\_small

Update run\_experiment.sh to accept a **workload binary path** as argument, so you can run:

./run\_experiment.sh ../workloads/parsec/blackscholes/.../blackscholes TournamentBP

or

./run\_experiment.sh ../workloads/mibench/automotive/basicmath/basicmath\_small LocalBP

## ✅ Recommended choice for assignment

* **Branch-heavy** → MiBench basicmath\_small
* **Compute-heavy** → PARSEC blackscholes

This matches exactly what your assignment wants: **one branch-heavy + one compute-heavy**.

👉 Do you want me to now **rewrite your** run\_all.sh so it loops over:

* both workloads (blackscholes, basicmath\_small)
* all branch predictors

and saves results in results/raw/<workload>/<BP>/?

chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ git clone https://github.com/bamos/parsec-benchmark.git ~/HPC-Projects/workloads/parsec Cloning into '/home/chirag/HPC-Projects/workloads/parsec'... remote: Enumerating objects: 20904, done. remote: Counting objects: 100% (10/10), done. remote: Compressing objects: 100% (10/10), done. remote: Total 20904 (delta 0), reused 2 (delta 0), pack-reused 20894 (from 1) Receiving objects: 100% (20904/20904), 49.74 MiB | 8.78 MiB/s, done. Resolving deltas: 100% (5647/5647), done. Updating files: 100% (20372/20372), done. chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ cd ~/HPC-Projects/workload/parsec -bash: cd: /home/chirag/HPC-Projects/workload/parsec: No such file or directory chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ ls README.md configs reports results scripts workloads chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ cd workloads/ chirag@chirag:~/HPC-Projects/A2-gem5-HPC/workloads$ ls chirag@chirag:~/HPC-Projects/A2-gem5-HPC/workloads$

###### ChatGPT said:

Nice 👍 you successfully cloned the **PARSEC benchmark**, but you accidentally typed the wrong folder name when trying to cd.

You wrote:

cd ~/HPC-Projects/workload/parsec

Notice: workload (singular).  
But your repo has workloads/ (plural).

## ✅ Correct path

Try:

cd ~/HPC-Projects/workloads/parsec

ls

You should now see all the PARSEC files (pkgs/, bin/, env.sh, etc.).

👉 Once you confirm you’re inside the right PARSEC folder, I’ll guide you through **building** blackscholes (your compute-heavy workload).

Do you want me to also set up the **MiBench clone** next so we can build basicmath\_small for the branch-heavy case?

##### You said:

chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ sudo apt update [sudo] password for chirag: Hit:1 https://cli.github.com/packages stable InRelease Hit:2 http://security.ubuntu.com/ubuntu noble-security InRelease Hit:3 http://archive.ubuntu.com/ubuntu noble InRelease Get:4 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB] Hit:5 http://archive.ubuntu.com/ubuntu noble-backports InRelease Get:6 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1390 kB] Get:7 http://archive.ubuntu.com/ubuntu noble-updates/universe amd64 Packages [1481 kB] Fetched 2997 kB in 59s (51.2 kB/s) Reading package lists... Done Building dependency tree... Done Reading state information... Done All packages are up to date. chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ sudo apt install -y gcc g++ make cmake unzip Reading package lists... Done Building dependency tree... Done Reading state information... Done gcc is already the newest version (4:13.2.0-7ubuntu1). gcc set to manually installed. g++ is already the newest version (4:13.2.0-7ubuntu1). g++ set to manually installed. make is already the newest version (4.3-4.1build2). make set to manually installed. The following package was automatically installed and is no longer required: libllvm19 Use 'sudo apt autoremove' to remove it. The following additional packages will be installed: cmake-data libarchive13t64 libjsoncpp25 librhash0 libuv1t64 Suggested packages: cmake-doc cmake-format elpa-cmake-mode ninja-build lrzip zip The following NEW packages will be installed: cmake cmake-data libarchive13t64 libjsoncpp25 librhash0 libuv1t64 unzip 0 upgraded, 7 newly installed, 0 to remove and 0 not upgraded. Need to get 14.2 MB of archives. After this operation, 50.7 MB of additional disk space will be used. Get:1 http://archive.ubuntu.com/ubuntu noble/main amd64 libuv1t64 amd64 1.48.0-1.1build1 [97.3 kB] Get:2 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 libarchive13t64 amd64 3.7.2-2ubuntu0.5 [382 kB] Get:3 http://archive.ubuntu.com/ubuntu noble/main amd64 libjsoncpp25 amd64 1.9.5-6build1 [82.8 kB] Get:4 http://archive.ubuntu.com/ubuntu noble/main amd64 librhash0 amd64 1.4.3-3build1 [129 kB] Get:5 http://archive.ubuntu.com/ubuntu noble/main amd64 cmake-data all 3.28.3-1build7 [2155 kB] Get:6 http://archive.ubuntu.com/ubuntu noble/main amd64 cmake amd64 3.28.3-1build7 [11.2 MB] Get:7 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 unzip amd64 6.0-28ubuntu4.1 [174 kB] Fetched 14.2 MB in 18s (790 kB/s) Selecting previously unselected package libuv1t64:amd64. (Reading database ... 86678 files and directories currently installed.) Preparing to unpack .../0-libuv1t64\_1.48.0-1.1build1\_amd64.deb ... Unpacking libuv1t64:amd64 (1.48.0-1.1build1) ... Selecting previously unselected package libarchive13t64:amd64. Preparing to unpack .../1-libarchive13t64\_3.7.2-2ubuntu0.5\_amd64.deb ... Unpacking libarchive13t64:amd64 (3.7.2-2ubuntu0.5) ... Selecting previously unselected package libjsoncpp25:amd64. Preparing to unpack .../2-libjsoncpp25\_1.9.5-6build1\_amd64.deb ... Unpacking libjsoncpp25:amd64 (1.9.5-6build1) ... Selecting previously unselected package librhash0:amd64. Preparing to unpack .../3-librhash0\_1.4.3-3build1\_amd64.deb ... Unpacking librhash0:amd64 (1.4.3-3build1) ... Selecting previously unselected package cmake-data. Preparing to unpack .../4-cmake-data\_3.28.3-1build7\_all.deb ... Unpacking cmake-data (3.28.3-1build7) ... Selecting previously unselected package cmake. Preparing to unpack .../5-cmake\_3.28.3-1build7\_amd64.deb ... Unpacking cmake (3.28.3-1build7) ... Selecting previously unselected package unzip. Preparing to unpack .../6-unzip\_6.0-28ubuntu4.1\_amd64.deb ... Unpacking unzip (6.0-28ubuntu4.1) ... Setting up unzip (6.0-28ubuntu4.1) ... Setting up libuv1t64:amd64 (1.48.0-1.1build1) ... Setting up libjsoncpp25:amd64 (1.9.5-6build1) ... Setting up librhash0:amd64 (1.4.3-3build1) ... Setting up cmake-data (3.28.3-1build7) ... Setting up libarchive13t64:amd64 (3.7.2-2ubuntu0.5) ... Setting up cmake (3.28.3-1build7) ... Processing triggers for man-db (2.12.0-4build2) ... Processing triggers for libc-bin (2.39-0ubuntu8.5) ... chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ git clone httos://github.com/bamos/parsec-benchmark.git ~/HPC-Projects/workload/parsec Cloning into '/home/chirag/HPC-Projects/workload/parsec'... git: 'remote-httos' is not a git command. See 'git --help'. The most similar command is remote-https chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ git clone https://github.com/bamos/parsec-benchmark.git ~/HPC-Projects/workloads/parsec Cloning into '/home/chirag/HPC-Projects/workloads/parsec'... remote: Enumerating objects: 20904, done. remote: Counting objects: 100% (10/10), done. remote: Compressing objects: 100% (10/10), done. remote: Total 20904 (delta 0), reused 2 (delta 0), pack-reused 20894 (from 1) Receiving objects: 100% (20904/20904), 49.74 MiB | 8.78 MiB/s, done. Resolving deltas: 100% (5647/5647), done. Updating files: 100% (20372/20372), done. chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ cd ~/HPC-Projects/workload/parsec -bash: cd: /home/chirag/HPC-Projects/workload/parsec: No such file or directory chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ ls README.md configs reports results scripts workloads chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ cd workloads/ chirag@chirag:~/HPC-Projects/A2-gem5-HPC/workloads$ ls chirag@chirag:~/HPC-Projects/A2-gem5-HPC/workloads$ cd ~/HPC-Projects/workloads/parsec ls CHANGELOG FAQ README bin env.sh man version CONTRIBUTORS LICENSE README.md config ext pkgs chirag@chirag:~/HPC-Projects/workloads/parsec$ source env.sh chirag@chirag:~/HPC-Projects/workloads/parsec$ parsecmgmt -a build -p blackscholes [PARSEC] Packages to build: parsec.blackscholes [PARSEC] [========== Building package parsec.blackscholes [1] ==========] [PARSEC] [---------- Analyzing package parsec.blackscholes ----------] [PARSEC] parsec.blackscholes does not depend on any other packages. [PARSEC] [---------- Building package parsec.blackscholes ----------] [PARSEC] Copying source code of package parsec.blackscholes. [PARSEC] Running 'env version=pthreads /usr/bin/make': rm -f blackscholes blackscholes.m4.cpp /usr/bin/m4 ./c.m4.pthreads blackscholes.c > blackscholes.m4.cpp /usr/bin/g++ -O3 -g -funroll-loops -fprefetch-loop-arrays -fpermissive -fno-exceptions -static-libgcc -Wl,--hash-style=both,--as-needed -DPARSEC\_VERSION=3.0-beta-20120904 -pthread -DENABLE\_THREADS -DNCO=4 blackscholes.m4.cpp -L/usr/lib64 -L/usr/lib -o blackscholes blackscholes.m4.cpp:408:16: warning: invalid suffix on literal; C++11 requires a space between literal and string macro [-Wliteral-suffix] 408 | printf("PARSEC Benchmark Suite Version "\_\_PARSEC\_XSTRING(PARSEC\_VERSION)"\n"); | ^ blackscholes.m4.cpp: In function ‘int main(int, char\*\*)’: blackscholes.m4.cpp:507:28: warning: format ‘%d’ expects argument of type ‘int’, but argument 2 has type ‘long unsigned int’ [-Wformat=] 507 | printf("Size of data: %d\n", numOptions \* (sizeof(OptionData) + sizeof(int))); | ~^ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ | | | | int long unsigned int | %ld [PARSEC] Running 'env version=pthreads /usr/bin/make install': mkdir -p /home/chirag/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin cp -f blackscholes /home/chirag/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes [PARSEC] [PARSEC] BIBLIOGRAPHY [PARSEC] [PARSEC] [1] Bienia. Benchmarking Modern Multiprocessors. Ph.D. Thesis, 2011. [PARSEC] [PARSEC] Done. chirag@chirag:~/HPC-Projects/workloads/parsec$ git clone https://github.com/embecosm/mibench.git ~/HPC-Projects/workloads/mibench Cloning into '/home/chirag/HPC-Projects/workloads/mibench'... remote: Enumerating objects: 3785, done. remote: Total 3785 (delta 0), reused 0 (delta 0), pack-reused 3785 (from 1) Receiving objects: 100% (3785/3785), 76.30 MiB | 9.07 MiB/s, done. Resolving deltas: 100% (558/558), done. chirag@chirag:~/HPC-Projects/workloads/parsec$ cd ~/HPC-Projects/workloads/mibench/automotive/basicmath make gcc -static -O3 basicmath\_small.c rad2deg.c cubic.c isqrt.c -o basicmath\_small -lm basicmath\_small.c: In function ‘main’: basicmath\_small.c:21:3: warning: implicit declaration of function ‘printf’ -Wimplicit-function-declaration] 21 | printf("\*\*\*\*\*\*\*\*\* CUBIC FUNCTIONS \*\*\*\*\*\*\*\*\*\*\*\n"); | ^~~~~~ basicmath\_small.c:2:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’ 1 | #include "snipmath.h" +++ |+#include <stdio.h> 2 | #include <math.h> basicmath\_small.c:21:3: warning: incompatible implicit declaration of built-in function ‘printf’ [-Wbuiltin-declaration-mismatch] 21 | printf("\*\*\*\*\*\*\*\*\* CUBIC FUNCTIONS \*\*\*\*\*\*\*\*\*\*\*\n"); | ^~~~~~ basicmath\_small.c:21:3: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’ basicmath\_small.c:78:3: warning: implicit declaration of function ‘puts’ [-Wimplicit-function-declaration] 78 | puts(""); | ^~~~ basicmath\_small.c:78:3: note: include ‘<stdio.h>’ or provide a declaration of ‘puts’ gcc -static -O3 basicmath\_large.c rad2deg.c cubic.c isqrt.c -o basicmath\_large -lm basicmath\_large.c: In function ‘main’: basicmath\_large.c:18:3: warning: implicit declaration of function ‘printf’ -Wimplicit-function-declaration] 18 | printf("\*\*\*\*\*\*\*\*\* CUBIC FUNCTIONS \*\*\*\*\*\*\*\*\*\*\*\n"); | ^~~~~~ basicmath\_large.c:2:1: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’ 1 | #include "snipmath.h" +++ |+#include <stdio.h> 2 | #include <math.h> basicmath\_large.c:18:3: warning: incompatible implicit declaration of built-in function ‘printf’ [-Wbuiltin-declaration-mismatch] 18 | printf("\*\*\*\*\*\*\*\*\* CUBIC FUNCTIONS \*\*\*\*\*\*\*\*\*\*\*\n"); | ^~~~~~ basicmath\_large.c:18:3: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’ basicmath\_large.c:116:3: warning: implicit declaration of function ‘puts’ [-Wimplicit-function-declaration] 116 | puts(""); | ^~~~ basicmath\_large.c:116:3: note: include ‘<stdio.h>’ or provide a declaration of ‘puts’ chirag@chirag:~/HPC-Projects/workloads/mibench/automotive/basicmath$ cd ~/HPC-Projects/workloads/mibench/automotive/basicmath chirag@chirag:~/HPC-Projects/workloads/mibench/automotive/basicmath$ make make: Nothing to be done for 'all'. chirag@chirag:~/HPC-Projects/workloads/mibench/automotive/basicmath$ cd .. chirag@chirag:~/HPC-Projects/workloads/mibench/automotive$ cd .. chirag@chirag:~/HPC-Projects/workloads/mibench$ cd .. chirag@chirag:~/HPC-Projects/workloads$ ls parsec/blackscholes/inst/amd64-linux.gcc/bin ls: cannot access 'parsec/blackscholes/inst/amd64-linux.gcc/bin': No such file or directory chirag@chirag:~/HPC-Projects/workloads$ cd parsec/ chirag@chirag:~/HPC-Projects/workloads/parsec$ ls CHANGELOG FAQ README bin env.sh log pkgs CONTRIBUTORS LICENSE README.md config ext man version chirag@chirag:~/HPC-Projects/workloads/parsec$

###### ChatGPT said:

Beautiful progress 🚀 — you’ve now got **both benchmark suites cloned and built**:

* ✅ **PARSEC**: blackscholes built successfully.
* ✅ **MiBench**: basicmath (small + large) built successfully.

The only confusion is **where blackscholes ended up** after building. Let’s fix that.

## 🔎 Where PARSEC puts binaries

By default, PARSEC installs binaries into:

pkgs/apps/<benchmark>/inst/amd64-linux.gcc/bin/

So for blackscholes, the binary should be:

~/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes

## ✅ Next steps

### 1. Check if blackscholes is there

ls ~/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin

You should see blackscholes.

### 2. Check MiBench basicmath

Binaries are in your build directory already:

~/HPC-Projects/workloads/mibench/automotive/basicmath/basicmath\_small

~/HPC-Projects/workloads/mibench/automotive/basicmath/basicmath\_large

### 3. Test them outside gem5 (to ensure they run natively)

~/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes 1 in\_10M.txt prices.txt

./workloads/mibench/automotive/basicmath/basicmath\_small

👉 If you don’t have PARSEC input files (in\_10M.txt), we can generate **dummy input** (small dataset) for testing inside gem5 — otherwise runs will be too long.

⚡ Once we confirm both binaries are runnable, I’ll show you how to:

* Modify your run\_all.sh so it loops over both **workloads** + **predictors**
* Save results in results/raw/<workload>/<predictor>/

Do you want me to show you how to create a **small dummy input file** for blackscholes (so you can run it fast in gem5)?

##### You said:

1. yes present 2. seems emplty chirag@chirag:~/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin$ cd ~ chirag@chirag:~$ ls A1-gem5-HPC HPC-Projects dummy chirag@chirag:~$ cd HPC-Projects/ chirag@chirag:~/HPC-Projects$ cd workload chirag@chirag:~/HPC-Projects/workload$ ls chirag@chirag:~/HPC-Projects/workload$

###### ChatGPT said:

Great 👍 thanks for checking. Let’s break this down carefully:

## ✅ What we know

1. **PARSEC**:  
   You confirmed blackscholes binary is present here:

~/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes

✔️ Perfect — we can run this inside gem5.

1. **MiBench**:  
   Your automotive/basicmath folder built without errors, but you’re not seeing the binaries. That usually means they were created but you’re looking in the wrong folder.

By default, make should produce:

* + basicmath\_small
  + basicmath\_large  
    inside the same directory where the sources are.

Path to check:

~/HPC-Projects/workloads/mibench/automotive/basicmath/

## 🔧 Let’s verify MiBench

Please run:

ls -lh ~/HPC-Projects/workloads/mibench/automotive/basicmath

You should see basicmath\_small and basicmath\_large as executables.  
If not, maybe the make failed silently — we’ll check logs.

## 🔧 Next Steps

* If binaries are there → ✅ we’re ready to integrate both workloads (blackscholes + basicmath\_small) into your pipeline.
* If missing → I’ll walk you through fixing the MiBench build (likely just needs #include <stdio.h> patch).

👉 Can you run that ls -lh ~/HPC-Projects/workloads/mibench/automotive/basicmath and tell me what shows up?

chirag@chirag:~/HPC-Projects/workloads/mibench/automotive/basicmath$ ls -lh ~/HPC-Projects/workloads/mibench/automotive/basicmath

total 1.8M

-rwxr-xr-x 1 chirag chirag 276 Sep 6 04:34 COMPILE

-rwxr-xr-x 1 chirag chirag 72 Sep 6 04:34 LICENSE

-rwxr-xr-x 1 chirag chirag 377 Sep 6 04:34 Makefile

-rwxr-xr-x 1 chirag chirag 887K Sep 6 04:34 basicmath\_large

-rwxr-xr-x 1 chirag chirag 3.2K Sep 6 04:34 basicmath\_large.c

-rwxr-xr-x 1 chirag chirag 887K Sep 6 04:34 basicmath\_small

-rwxr-xr-x 1 chirag chirag 2.3K Sep 6 04:34 basicmath\_small.c

-rwxr-xr-x 1 chirag chirag 1.6K Sep 6 04:34 cubic.c

-rwxr-xr-x 1 chirag chirag 2.7K Sep 6 04:34 isqrt.c

-rwxr-xr-x 1 chirag chirag 219 Sep 6 04:34 pi.h

-rwxr-xr-x 1 chirag chirag 763 Sep 6 04:34 rad2deg.c

-rwxr-xr-x 1 chirag chirag 924 Sep 6 04:34 round.h

-rwxr-xr-x 1 chirag chirag 45 Sep 6 04:34 runme\_large.sh

-rwxr-xr-x 1 chirag chirag 45 Sep 6 04:34 runme\_small.sh

-rwxr-xr-x 1 chirag chirag 3.0K Sep 6 04:34 snipmath.h

-rwxr-xr-x 1 chirag chirag 956 Sep 6 04:34 sniptype.h

[chirag@chirag](mailto:chirag@chirag):~/HPC-Projects/workloads/mibench/automotive/basicmath$  
  
  
Perfect 🎉 — confirmed:

* **PARSEC blackscholes** binary is ready:  
  ~/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes
* **MiBench basicmath** binaries are ready:  
  ~/HPC-Projects/workloads/mibench/automotive/basicmath/basicmath\_small  
  ~/HPC-Projects/workloads/mibench/automotive/basicmath/basicmath\_large

That means you now have:  
✅ one **compute-heavy workload** (blackscholes)  
✅ one **branch-heavy workload** (basicmath\_small/large)

Exactly what your assignment asked for

Assignment-Branch

.

### 🔧 Next step: integrate workloads into your pipeline

We’ll modify your run\_all.sh so it loops over both **workloads** and **branch predictors**, saving results into this structure:

results/raw/<workload>/<predictor>/

So you’ll get things like:

results/raw/blackscholes/TournamentBP/stats.txt

results/raw/basicmath\_small/LocalBP/stats.txt

👉 Do you want me to go ahead and **rewrite** run\_all.sh **+ update parser** so it supports multiple workloads automatically? That way, you can just run:

./run\_all.sh

and it will produce CSV + plots across both **blackscholes** and **basicmath\_small**.

Sources

##### You said:

sure lets do it but before that lets puch code to github like we were doing before

###### ChatGPT said:

Good call 👍 — always commit before a big change.  
Let’s push your current scripts + results to GitHub.

### 🔧 Step 1. Check repo status

From your repo root (~/HPC-Projects/A2-gem5-HPC):

cd ~/HPC-Projects/A2-gem5-HPC

git status

This shows which files are modified or untracked.

### 🔧 Step 2. Add & commit

If you see new files (scripts, results, workloads .gitkeep, etc.), stage them:

git add scripts/\*.sh scripts/\*.py results/ workloads/.gitkeep

git commit -m "chore: add workloads setup and working scripts (before multi-workload update)"

### 🔧 Step 3. Push to GitHub

git push

Now your teammate + future self have a safe checkpoint ✅.

👉 After this, I’ll help you **rewrite** run\_all.sh **+ parser** for multi-workload support.  
Do you want me to also add a .gitignore (so huge PARSEC/MiBench folders don’t get pushed accidentally)?

##### You said:

chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ git add scripts/\*.sh scripts/\*.py results/ workloads/.gitkeep fatal: pathspec 'workloads/.gitkeep' did not match any files chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ git add scripts/\*.sh scripts/\*.py results/ workloads/.gitkeep git commit -m "chore: add workloads setup and working scripts (before multi-workload update)" fatal: pathspec 'workloads/.gitkeep' did not match any files On branch main Your branch is up to date with 'origin/main'. Untracked files: (use "git add <file>..." to include in what will be committed) results/summary/ nothing added to commit but untracked files present (use "git add" to track) chirag@chirag:~/HPC-Projects/A2-gem5-HPC$

###### ChatGPT said:

Ahh I see what happened 👍

* You don’t actually have a workloads/.gitkeep file yet → so Git complained.
* Also, results/summary/ is untracked, so your CSV + plots aren’t committed either.

Let’s fix this properly:

### 🔧 Step 1. Add a .gitkeep inside workloads

This ensures Git tracks the folder but not the huge benchmarks:

touch workloads/.gitkeep

(We’ll also add a .gitignore to avoid committing PARSEC/MiBench binaries accidentally.)

### 🔧 Step 2. Create/update .gitignore

Inside your repo root:

nano .gitignore

Paste this:

# Ignore benchmark binaries and large data

workloads/parsec/

workloads/mibench/

# Ignore gem5 build outputs

m5out/

\*.log

# Ignore generated results, but keep summaries

results/raw/

!results/summary/

Save + exit.

### 🔧 Step 3. Stage & commit

Now:

git add scripts/\*.sh scripts/\*.py results/summary workloads/.gitkeep .gitignore

git commit -m "chore: add workloads placeholder, results summary, and scripts (before multi-workload update)"

git push

✅ This way:

* **Scripts** are tracked
* **Results summaries** are tracked (CSV + plots)
* **Workloads** (big benchmarks) stay local

👉 Want me to also set up a README\_workloads.md in your repo that explains how to clone & build PARSEC/MiBench (so your teammate/prof can reproduce)?

##### You said:

can we update readme?

###### ChatGPT said:

Absolutely 👍 Updating the README.md is a great idea so your repo is self-explanatory for your teammate (and examiner).

Here’s what I suggest adding — we’ll extend your existing README.md with a **Workloads Setup** section.

### ✍️ Suggested README Update

# A2-gem5-HPC — Branch Prediction Experiments

## Overview

This project evaluates different branch prediction schemes in gem5 using an Out-of-Order CPU (O3CPU).

We automate experiments, parse results into CSV, and generate plots for IPC and misprediction rate.

---

## Repo Structure

scripts/ # Automation scripts (run\_all.sh, run\_experiment.sh, parse\_stats.py, plot\_results.py)  
results/  
raw/ # Raw stats.txt outputs (ignored in git)  
summary/ # Processed CSV + plots (tracked)  
workloads/ # Benchmark sources (not committed, see setup below)

---

## Workloads Setup

We use two representative benchmarks:

- \*\*PARSEC blackscholes\*\* (compute-heavy)

- \*\*MiBench basicmath\_small\*\* (branch-heavy)

### 1. Install dependencies

```bash

sudo apt update

sudo apt install -y gcc g++ make cmake unzip

### 2. Clone and build PARSEC

git clone https://github.com/bamos/parsec-benchmark.git ~/HPC-Projects/workloads/parsec

cd ~/HPC-Projects/workloads/parsec

source env.sh

parsecmgmt -a build -p blackscholes

Binary will be at:

~/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes

### 3. Clone and build MiBench

git clone https://github.com/embecosm/mibench.git ~/HPC-Projects/workloads/mibench

cd ~/HPC-Projects/workloads/mibench/automotive/basicmath

make

Binaries will be at:

basicmath\_small

basicmath\_large

## Running Experiments

Run everything (all predictors + workloads):

cd scripts

./run\_all.sh

Outputs:

* results/summary/results.csv → IPC + misprediction rates
* results/summary/ipc\_comparison.png → IPC plot
* results/summary/mispred\_rate\_comparison.png → Misprediction plot

## Notes

* workloads/ is not tracked in Git (see .gitignore). Each user must set it up locally.
* Use run\_experiment.sh for individual runs if needed.

---

👉 Do you want me to \*\*append this into your existing `README.md` automatically\*\*, or show you how to do it manually?

##### You said:

here is how it was done before # A2-gem5-HPC — Branch Prediction Experiments ## Overview This project evaluates different branch prediction schemes in gem5 using an Out-of-Order CPU (O3CPU). We automate experiments, parse results into CSV, and generate plots for IPC and misprediction rate. --- ## Repo Structure

###### ChatGPT said:

Got it 👍 thanks for showing me the current style of your README.md.  
We’ll keep that format and extend it cleanly.

Here’s how the updated README.md will look (continuing after **Repo Structure**):

# A2-gem5-HPC — Branch Prediction Experiments

## Overview

This project evaluates different branch prediction schemes in gem5 using an Out-of-Order CPU (O3CPU).

We automate experiments, parse results into CSV, and generate plots for IPC and misprediction rate.

---

## Repo Structure

scripts/ # Automation scripts (run\_all.sh, run\_experiment.sh, parse\_stats.py, plot\_results.py)  
results/  
raw/ # Raw stats.txt outputs (ignored in git)  
summary/ # Processed CSV + plots (tracked)  
workloads/ # Benchmark sources (not committed, see setup below)

---

## Workloads Setup

We use two representative benchmarks:

- \*\*PARSEC blackscholes\*\* (compute-heavy)

- \*\*MiBench basicmath\_small\*\* (branch-heavy)

### 1. Install dependencies

```bash

sudo apt update

sudo apt install -y gcc g++ make cmake unzip

### 2. Clone and build PARSEC

git clone https://github.com/bamos/parsec-benchmark.git ~/HPC-Projects/workloads/parsec

cd ~/HPC-Projects/workloads/parsec

source env.sh

parsecmgmt -a build -p blackscholes

Binary will be at:

~/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes

### 3. Clone and build MiBench

git clone https://github.com/embecosm/mibench.git ~/HPC-Projects/workloads/mibench

cd ~/HPC-Projects/workloads/mibench/automotive/basicmath

make

Binaries will be at:

basicmath\_small

basicmath\_large

## Running Experiments

Run the full pipeline (all predictors + workloads):

cd scripts

./run\_all.sh

Outputs:

* results/summary/results.csv → IPC + misprediction rates
* results/summary/ipc\_comparison.png → IPC plot
* results/summary/mispred\_rate\_comparison.png → Misprediction plot

## Notes

* workloads/ is not tracked in Git (see .gitignore). Each user must set it up locally.
* Use run\_experiment.sh for individual runs if needed.

---

👉 Do you want me to \*\*append this automatically to your README.md\*\* with a git commit, or would you prefer to open the file in `nano` and paste this in manually?

##### You said:

I want in this structure ### 2025-09-06 (Step 6) - \*\*Setup:\*\* Created run\_all.sh master script to automate the full pipeli>- \*\*Experiment:\*\* Running ./scripts/run\_all.sh executes: 1. Batch gem5 runs with all branch predictors 2. Parsing of stats into results/summary/results.csv 3. Plot generation (ipc\_comparison.png, mispred\_rate\_comparison.png) - \*\*Results:\*\* One-command reproducibility achieved. - \*\*Notes:\*\* This script is the main entry point for running the entire pro>

###### ChatGPT said:

Perfect 👌 you’re keeping a **changelog style** inside README.md.  
Let’s extend it with a new dated step for **Workloads Setup** (Step 7).

Here’s how it will look:

### 2025-09-06 (Step 6)

- \*\*Setup:\*\* Created `run\_all.sh` master script to automate the full pipeline

- \*\*Experiment:\*\* Running `./scripts/run\_all.sh` executes:

1. Batch gem5 runs with all branch predictors

2. Parsing of stats into `results/summary/results.csv`

3. Plot generation (`ipc\_comparison.png`, `mispred\_rate\_comparison.png`)

- \*\*Results:\*\* One-command reproducibility achieved.

- \*\*Notes:\*\* This script is the main entry point for running the entire project.

---

### 2025-09-06 (Step 7)

- \*\*Setup:\*\* Added real workloads (PARSEC + MiBench) to make experiments realistic.

- \*\*Workloads:\*\*

- \*\*PARSEC blackscholes\*\* → compute-heavy benchmark

Build steps:

```bash

git clone https://github.com/bamos/parsec-benchmark.git ~/HPC-Projects/workloads/parsec

cd ~/HPC-Projects/workloads/parsec

source env.sh

parsecmgmt -a build -p blackscholes

```

Binary at:

```

~/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes

```

- \*\*MiBench basicmath\_small\*\* → branch-heavy benchmark

Build steps:

```bash

git clone https://github.com/embecosm/mibench.git ~/HPC-Projects/workloads/mibench

cd ~/HPC-Projects/workloads/mibench/automotive/basicmath

make

```

Binaries at:

```

basicmath\_small

basicmath\_large

```

- \*\*Experiment:\*\* These workloads will be integrated into `run\_all.sh` so that results are saved under:

results/raw/<workload>/<predictor>/

- \*\*Results:\*\* Prepared realistic benchmarks for IPC + misprediction evaluation.

- \*\*Notes:\*\* `workloads/` folder is ignored in Git. Each user must set up workloads locally by following these steps.

👉 Do you want me to **append this Step 7 block automatically to your README.md** and commit, or should I just leave it here for you to paste manually?

##### You said:

okay this is done whats next?

###### ChatGPT said:

Nice 🚀 — now your repo + README are clean and documented.  
The **next step (Step 8)** is what we talked about earlier:

👉 **Update your pipeline (**run\_all.sh **+ parser) to handle multiple workloads automatically.**

Right now:

* run\_all.sh → loops over predictors only.
* parse\_stats.py → assumes 1 stats.txt per predictor.

We need to upgrade them so they produce structured results like this:

results/raw/blackscholes/TournamentBP/stats.txt

results/raw/blackscholes/LocalBP/stats.txt

results/raw/basicmath\_small/TournamentBP/stats.txt

results/raw/basicmath\_small/LocalBP/stats.txt

...

And the CSV will look like:

Workload,BranchPredictor,IPC,MispredRate

blackscholes,TournamentBP,1.23,0.04

blackscholes,LocalBP,1.10,0.08

basicmath\_small,TournamentBP,1.30,0.02

basicmath\_small,LocalBP,1.05,0.09

### 🔧 Step 8 Plan

1. **Edit** run\_experiment.sh → accept both workload path + predictor name.
2. **Update** run\_all.sh → loop over all workloads and predictors.
3. **Update** parse\_stats.py → parse results into a 4-column CSV (Workload + Predictor + IPC + MispredRate).
4. **Update** plot\_results.py → group plots per workload.

👉 Do you want me to start by rewriting run\_all.sh for **multi-workload support first**, or update parse\_stats.py at the same time so you can immediately test the new CSV?

##### You said:

lets do step 8

###### ChatGPT said:

Perfect 👌 let’s roll into **Step 8: multi-workload pipeline**.

We’ll make changes in **three scripts**:

## 1️⃣ Update run\_experiment.sh

Right now it only takes predictor name. Let’s make it take **two arguments**: workload path + predictor.

#!/bin/bash

# run\_experiment.sh

# Run gem5 O3CPU with a given workload + branch predictor

# Usage: ./run\_experiment.sh <workload> <BP>

WORKLOAD=$1 # full path to binary

BP=$2 # branch predictor (e.g., TournamentBP, LocalBP, BiModeBP, LTage)

# Derive workload name (strip path)

WORKLOAD\_NAME=$(basename $WORKLOAD)

OUTDIR="../results/raw/${WORKLOAD\_NAME}/${BP}"

# Paths

GEM5=~/HPC-Projects/gem5/build/X86/gem5.opt

SE=~/HPC-Projects/gem5/configs/deprecated/example/se.py

mkdir -p $OUTDIR

$GEM5 \

--outdir=$OUTDIR \

$SE \

-c $WORKLOAD \

--cpu-type=O3CPU \

--bp-type=$BP \

--caches --l2cache

## 2️⃣ Update run\_all.sh

Now loop over workloads **and** predictors.

#!/bin/bash

# run\_all.sh

# Full pipeline: run predictors -> parse stats -> plot results

# Workloads

WORKLOADS=(

"../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes"

"../workloads/mibench/automotive/basicmath/basicmath\_small"

)

# Predictors

BPS=("TournamentBP" "LocalBP" "BiModeBP" "LTage")

echo ">>> [1/3] Running gem5 experiments..."

for WL in "${WORKLOADS[@]}"; do

for BP in "${BPS[@]}"; do

echo ">>> Running $WL with $BP"

./run\_experiment.sh $WL $BP

done

done

echo ">>> [2/3] Parsing stats into CSV..."

./parse\_stats.py

echo ">>> [3/3] Generating plots..."

./plot\_results.py

echo "✅ All steps complete. Check results in ../results/summary/"

## 3️⃣ Update parse\_stats.py

Now include workload name in CSV.

#!/usr/bin/env python3

import os, csv

RAW\_DIR = "../results/raw"

OUT\_FILE = "../results/summary/results.csv"

def extract\_stats(stats\_file):

ipc, lookups, mispred = None, 0, 0

with open(stats\_file) as f:

for line in f:

if "sim\_ipc" in line or "system.cpu.ipc" in line:

ipc = float(line.split()[1])

if "branchPred.lookups\_0::total" in line:

lookups = int(line.split()[1])

if "branchPred.squashes\_0::total" in line:

mispred = int(line.split()[1])

mispred\_rate = (mispred / lookups) if lookups > 0 else 0

return ipc, mispred\_rate

def main():

rows = []

for workload in os.listdir(RAW\_DIR):

wl\_path = os.path.join(RAW\_DIR, workload)

if os.path.isdir(wl\_path):

for bp in os.listdir(wl\_path):

stats\_file = os.path.join(wl\_path, bp, "stats.txt")

if os.path.isfile(stats\_file):

ipc, mispred\_rate = extract\_stats(stats\_file)

rows.append((workload, bp, ipc, mispred\_rate))

with open(OUT\_FILE, "w", newline="") as f:

writer = csv.writer(f)

writer.writerow(["Workload", "BranchPredictor", "IPC", "MispredRate"])

writer.writerows(rows)

print(f"✅ Results written to {OUT\_FILE}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

## 4️⃣ Update plot\_results.py

We’ll make separate plots **per workload**.

#!/usr/bin/env python3

import pandas as pd

import matplotlib.pyplot as plt

import os

CSV\_FILE = "../results/summary/results.csv"

OUT\_DIR = "../results/summary"

df = pd.read\_csv(CSV\_FILE)

# Group by workload

for wl, group in df.groupby("Workload"):

# IPC Plot

plt.figure(figsize=(6,4))

plt.bar(group["BranchPredictor"], group["IPC"])

plt.title(f"IPC for {wl}")

plt.ylabel("IPC")

plt.xlabel("Branch Predictor")

plt.tight\_layout()

plt.savefig(os.path.join(OUT\_DIR, f"{wl}\_ipc.png"))

plt.close()

# Misprediction Plot

plt.figure(figsize=(6,4))

plt.bar(group["BranchPredictor"], group["MispredRate"])

plt.title(f"Misprediction Rate for {wl}")

plt.ylabel("Misprediction Rate")

plt.xlabel("Branch Predictor")

plt.tight\_layout()

plt.savefig(os.path.join(OUT\_DIR, f"{wl}\_mispred.png"))

plt.close()

print("✅ Plots saved in results/summary/")

### ✅ End of Step 8

Now your workflow does:

cd scripts

./run\_all.sh

It will produce:

* results/summary/results.csv
* Plots per workload:
  + blackscholes\_ipc.png
  + blackscholes\_mispred.png
  + basicmath\_small\_ipc.png
  + basicmath\_small\_mispred.png

👉 Do you want me to also update your README.md with a new **Step 8 changelog entry** describing this multi-workload support?

Perfect 👌 you’re keeping a **changelog style** inside README.md.  
Let’s extend it with a new dated step for **Workloads Setup** (Step 7).

Here’s how it will look:

### 2025-09-06 (Step 6)

- \*\*Setup:\*\* Created `run\_all.sh` master script to automate the full pipeline

- \*\*Experiment:\*\* Running `./scripts/run\_all.sh` executes:

1. Batch gem5 runs with all branch predictors

2. Parsing of stats into `results/summary/results.csv`

3. Plot generation (`ipc\_comparison.png`, `mispred\_rate\_comparison.png`)

- \*\*Results:\*\* One-command reproducibility achieved.

- \*\*Notes:\*\* This script is the main entry point for running the entire project.

---

### 2025-09-06 (Step 7)

- \*\*Setup:\*\* Added real workloads (PARSEC + MiBench) to make experiments realistic.

- \*\*Workloads:\*\*

- \*\*PARSEC blackscholes\*\* → compute-heavy benchmark

Build steps:

```bash

git clone https://github.com/bamos/parsec-benchmark.git ~/HPC-Projects/workloads/parsec

cd ~/HPC-Projects/workloads/parsec

source env.sh

parsecmgmt -a build -p blackscholes

```

Binary at:

```

~/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes

```

- \*\*MiBench basicmath\_small\*\* → branch-heavy benchmark

Build steps:

```bash

git clone https://github.com/embecosm/mibench.git ~/HPC-Projects/workloads/mibench

cd ~/HPC-Projects/workloads/mibench/automotive/basicmath

make

```

Binaries at:

```

basicmath\_small

basicmath\_large

```

- \*\*Experiment:\*\* These workloads will be integrated into `run\_all.sh` so that results are saved under:

results/raw/<workload>/<predictor>/

- \*\*Results:\*\* Prepared realistic benchmarks for IPC + misprediction evaluation.

- \*\*Notes:\*\* `workloads/` folder is ignored in Git. Each user must set up workloads locally by following these steps.

👉 Do you want me to **append this Step 7 block automatically to your README.md** and commit, or should I just leave it here for you to paste manually?

##### You said:

okay this is done whats next?

###### ChatGPT said:

Nice 🚀 — now your repo + README are clean and documented.  
The **next step (Step 8)** is what we talked about earlier:

👉 **Update your pipeline (**run\_all.sh **+ parser) to handle multiple workloads automatically.**

Right now:

* run\_all.sh → loops over predictors only.
* parse\_stats.py → assumes 1 stats.txt per predictor.

We need to upgrade them so they produce structured results like this:

results/raw/blackscholes/TournamentBP/stats.txt

results/raw/blackscholes/LocalBP/stats.txt

results/raw/basicmath\_small/TournamentBP/stats.txt

results/raw/basicmath\_small/LocalBP/stats.txt

...

And the CSV will look like:

Workload,BranchPredictor,IPC,MispredRate

blackscholes,TournamentBP,1.23,0.04

blackscholes,LocalBP,1.10,0.08

basicmath\_small,TournamentBP,1.30,0.02

basicmath\_small,LocalBP,1.05,0.09

### 🔧 Step 8 Plan

1. **Edit** run\_experiment.sh → accept both workload path + predictor name.
2. **Update** run\_all.sh → loop over all workloads and predictors.
3. **Update** parse\_stats.py → parse results into a 4-column CSV (Workload + Predictor + IPC + MispredRate).
4. **Update** plot\_results.py → group plots per workload.

👉 Do you want me to start by rewriting run\_all.sh for **multi-workload support first**, or update parse\_stats.py at the same time so you can immediately test the new CSV?

##### You said:

lets do step 8

###### ChatGPT said:

Perfect 👌 let’s roll into **Step 8: multi-workload pipeline**.

We’ll make changes in **three scripts**:

## 1️⃣ Update run\_experiment.sh

Right now it only takes predictor name. Let’s make it take **two arguments**: workload path + predictor.

#!/bin/bash

# run\_experiment.sh

# Run gem5 O3CPU with a given workload + branch predictor

# Usage: ./run\_experiment.sh <workload> <BP>

WORKLOAD=$1 # full path to binary

BP=$2 # branch predictor (e.g., TournamentBP, LocalBP, BiModeBP, LTage)

# Derive workload name (strip path)

WORKLOAD\_NAME=$(basename $WORKLOAD)

OUTDIR="../results/raw/${WORKLOAD\_NAME}/${BP}"

# Paths

GEM5=~/HPC-Projects/gem5/build/X86/gem5.opt

SE=~/HPC-Projects/gem5/configs/deprecated/example/se.py

mkdir -p $OUTDIR

$GEM5 \

--outdir=$OUTDIR \

$SE \

-c $WORKLOAD \

--cpu-type=O3CPU \

--bp-type=$BP \

--caches --l2cache

## 2️⃣ Update run\_all.sh

Now loop over workloads **and** predictors.

#!/bin/bash

# run\_all.sh

# Full pipeline: run predictors -> parse stats -> plot results

# Workloads

WORKLOADS=(

"../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes"

"../workloads/mibench/automotive/basicmath/basicmath\_small"

)

# Predictors

BPS=("TournamentBP" "LocalBP" "BiModeBP" "LTage")

echo ">>> [1/3] Running gem5 experiments..."

for WL in "${WORKLOADS[@]}"; do

for BP in "${BPS[@]}"; do

echo ">>> Running $WL with $BP"

./run\_experiment.sh $WL $BP

done

done

echo ">>> [2/3] Parsing stats into CSV..."

./parse\_stats.py

echo ">>> [3/3] Generating plots..."

./plot\_results.py

echo "✅ All steps complete. Check results in ../results/summary/"

## 3️⃣ Update parse\_stats.py

Now include workload name in CSV.

#!/usr/bin/env python3

import os, csv

RAW\_DIR = "../results/raw"

OUT\_FILE = "../results/summary/results.csv"

def extract\_stats(stats\_file):

ipc, lookups, mispred = None, 0, 0

with open(stats\_file) as f:

for line in f:

if "sim\_ipc" in line or "system.cpu.ipc" in line:

ipc = float(line.split()[1])

if "branchPred.lookups\_0::total" in line:

lookups = int(line.split()[1])

if "branchPred.squashes\_0::total" in line:

mispred = int(line.split()[1])

mispred\_rate = (mispred / lookups) if lookups > 0 else 0

return ipc, mispred\_rate

def main():

rows = []

for workload in os.listdir(RAW\_DIR):

wl\_path = os.path.join(RAW\_DIR, workload)

if os.path.isdir(wl\_path):

for bp in os.listdir(wl\_path):

stats\_file = os.path.join(wl\_path, bp, "stats.txt")

if os.path.isfile(stats\_file):

ipc, mispred\_rate = extract\_stats(stats\_file)

rows.append((workload, bp, ipc, mispred\_rate))

with open(OUT\_FILE, "w", newline="") as f:

writer = csv.writer(f)

writer.writerow(["Workload", "BranchPredictor", "IPC", "MispredRate"])

writer.writerows(rows)

print(f"✅ Results written to {OUT\_FILE}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

## 4️⃣ Update plot\_results.py

We’ll make separate plots **per workload**.

#!/usr/bin/env python3

import pandas as pd

import matplotlib.pyplot as plt

import os

CSV\_FILE = "../results/summary/results.csv"

OUT\_DIR = "../results/summary"

df = pd.read\_csv(CSV\_FILE)

# Group by workload

for wl, group in df.groupby("Workload"):

# IPC Plot

plt.figure(figsize=(6,4))

plt.bar(group["BranchPredictor"], group["IPC"])

plt.title(f"IPC for {wl}")

plt.ylabel("IPC")

plt.xlabel("Branch Predictor")

plt.tight\_layout()

plt.savefig(os.path.join(OUT\_DIR, f"{wl}\_ipc.png"))

plt.close()

# Misprediction Plot

plt.figure(figsize=(6,4))

plt.bar(group["BranchPredictor"], group["MispredRate"])

plt.title(f"Misprediction Rate for {wl}")

plt.ylabel("Misprediction Rate")

plt.xlabel("Branch Predictor")

plt.tight\_layout()

plt.savefig(os.path.join(OUT\_DIR, f"{wl}\_mispred.png"))

plt.close()

print("✅ Plots saved in results/summary/")

### ✅ End of Step 8

Now your workflow does:

cd scripts

./run\_all.sh

It will produce:

* results/summary/results.csv
* Plots per workload:
  + blackscholes\_ipc.png
  + blackscholes\_mispred.png
  + basicmath\_small\_ipc.png
  + basicmath\_small\_mispred.png

👉 Do you want me to also update your README.md with a new **Step 8 changelog entry** describing this multi-workload support?

##### You said:

where can I see these images that got generated

###### ChatGPT said:

Good question 👍

Since you’re on **WSL (Ubuntu on Windows)**, there are a few ways to view the plots that plot\_results.py generated inside results/summary/.

### 🔎 Option 1: Open in Windows File Explorer (easiest)

Run this from WSL:

explorer.exe .

If you’re in ~/HPC-Projects/A2-gem5-HPC/results/summary, this will open that folder in Windows Explorer, and you can just double-click blackscholes\_ipc.png, etc.

### 🔎 Option 2: Use an image viewer in WSL

Install feh or eog:

sudo apt install -y feh

feh ../results/summary/blackscholes\_ipc.png

But note: this only works if you have X11/GUI support configured (most WSL setups don’t unless you installed an X server like VcXsrv).

### 🔎 Option 3: Open directly in VS Code

If you’re using VS Code in WSL mode, just browse to the results/summary/ folder → click the PNGs → they open in VS Code’s image viewer.

✅ My recommendation: **Option 1 (**explorer.exe .**)** — super simple, no extra setup.

👉 Do you want me to also modify plot\_results.py so it saves one **combined PDF with all plots**, so you can drop it straight into your report?

##### You said:

chirag@chirag:~/HPC-Projects/A2-gem5-HPC/scripts$ ls parse\_stats.py plot\_results.py run\_all.sh run\_experiment.sh chirag@chirag:~/HPC-Projects/A2-gem5-HPC/scripts$ cd .. chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ ls README.md configs reports results scripts workloads chirag@chirag:~/HPC-Projects/A2-gem5-HPC$ cd results/ chirag@chirag:~/HPC-Projects/A2-gem5-HPC/results$ cd summary/ chirag@chirag:~/HPC-Projects/A2-gem5-HPC/results/summary$ ls ipc\_comparison.png mispred\_rate\_comparison.png results.csv chirag@chirag:~/HPC-Projects/A2-gem5-HPC/results/summary$ cd results.csv -bash: cd: results.csv: Not a directory chirag@chirag:~/HPC-Projects/A2-gem5-HPC/results/summary$ nano results.csv chirag@chirag:~/HPC-Projects/A2-gem5-HPC/results/summary$

###### ChatGPT said:

Nice 👍 you navigated correctly into results/summary/.

Here’s what you’ve got there:

ipc\_comparison.png # old plot (before Step 8, only predictors)

mispred\_rate\_comparison.png # old plot (before Step 8, only predictors)

results.csv # your latest summary table

That explains why you don’t yet see the **per-workload images** (blackscholes\_ipc.png, etc.).  
You probably haven’t re-run the **new Step 8 scripts** yet after updating them.

### ✅ To generate the new plots:

From scripts/ folder, run:

cd ~/HPC-Projects/A2-gem5-HPC/scripts

./run\_all.sh

This will:

1. Run gem5 for each workload + predictor
2. Parse stats into results/summary/results.csv
3. Generate **new plots**:
   * blackscholes\_ipc.png
   * blackscholes\_mispred.png
   * basicmath\_small\_ipc.png
   * basicmath\_small\_mispred.png

### 🔎 To view them

* From inside results/summary/, run:

explorer.exe .

→ Opens the folder in Windows File Explorer, so you can double-click PNGs.

👉 Want me to also modify plot\_results.py so that it **keeps the old combined plots** (ipc\_comparison.png, mispred\_rate\_comparison.png) and produces the new per-workload plots? That way you’ll have both global + workload-specific views.

##### You said:

chirag@chirag:~/HPC-Projects/A2-gem5-HPC/results/summary$ ls ipc\_comparison.png mispred\_rate\_comparison.png results.csv chirag@chirag:~/HPC-Projects/A2-gem5-HPC/results/summary$ cd results.csv -bash: cd: results.csv: Not a directory chirag@chirag:~/HPC-Projects/A2-gem5-HPC/results/summary$ nano results.csv chirag@chirag:~/HPC-Projects/A2-gem5-HPC/results/summary$ cd ~/HPC-Projects/A2-gem5-HPC/scripts chirag@chirag:~/HPC-Projects/A2-gem5-HPC/scripts$ ./run\_all.sh >>> [1/3] Running gem5 experiments... >>> Running ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes with TournamentBP gem5 Simulator System. https://www.gem5.org gem5 is copyrighted software; use the --copyright option for details. gem5 version 25.0.0.1 gem5 compiled Sep 6 2025 02:38:56 gem5 started Sep 6 2025 04:59:09 gem5 executing on chirag, pid 24096 command line: /home/chirag/HPC-Projects/gem5/build/X86/gem5.opt --outdir=../results/raw/blackscholes/TournamentBP /home/chirag/HPC-Projects/gem5/configs/deprecated/example/se.py -c ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes --cpu-type=O3CPU --bp-type=TournamentBP --caches --l2cache warn: The se.py script is deprecated. It will be removed in future releases of gem5. src/base/loader/image\_file\_data.cc:105: fatal: fatal condition fd < 0 occurred: Failed to open file ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes. This error typically occurs when the file path specified is incorrect. Memory Usage: 123996 KBytes >>> Running ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes with LocalBP gem5 Simulator System. https://www.gem5.org gem5 is copyrighted software; use the --copyright option for details. gem5 version 25.0.0.1 gem5 compiled Sep 6 2025 02:38:56 gem5 started Sep 6 2025 04:59:09 gem5 executing on chirag, pid 24100 command line: /home/chirag/HPC-Projects/gem5/build/X86/gem5.opt --outdir=../results/raw/blackscholes/LocalBP /home/chirag/HPC-Projects/gem5/configs/deprecated/example/se.py -c ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes --cpu-type=O3CPU --bp-type=LocalBP --caches --l2cache warn: The se.py script is deprecated. It will be removed in future releases of gem5. src/base/loader/image\_file\_data.cc:105: fatal: fatal condition fd < 0 occurred: Failed to open file ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes. This error typically occurs when the file path specified is incorrect. Memory Usage: 124000 KBytes >>> Running ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes with BiModeBP gem5 Simulator System. https://www.gem5.org gem5 is copyrighted software; use the --copyright option for details. gem5 version 25.0.0.1 gem5 compiled Sep 6 2025 02:38:56 gem5 started Sep 6 2025 04:59:10 gem5 executing on chirag, pid 24104 command line: /home/chirag/HPC-Projects/gem5/build/X86/gem5.opt --outdir=../results/raw/blackscholes/BiModeBP /home/chirag/HPC-Projects/gem5/configs/deprecated/example/se.py -c ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes --cpu-type=O3CPU --bp-type=BiModeBP --caches --l2cache warn: The se.py script is deprecated. It will be removed in future releases of gem5. src/base/loader/image\_file\_data.cc:105: fatal: fatal condition fd < 0 occurred: Failed to open file ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes. This error typically occurs when the file path specified is incorrect. Memory Usage: 123996 KBytes >>> Running ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes with LTage gem5 Simulator System. https://www.gem5.org gem5 is copyrighted software; use the --copyright option for details. gem5 version 25.0.0.1 gem5 compiled Sep 6 2025 02:38:56 gem5 started Sep 6 2025 04:59:10 gem5 executing on chirag, pid 24108 command line: /home/chirag/HPC-Projects/gem5/build/X86/gem5.opt --outdir=../results/raw/blackscholes/LTage /home/chirag/HPC-Projects/gem5/configs/deprecated/example/se.py -c ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes --cpu-type=O3CPU --bp-type=LTage --caches --l2cache warn: The se.py script is deprecated. It will be removed in future releases of gem5. usage: se.py [-h] [-n NUM\_CPUS] [--sys-voltage SYS\_VOLTAGE] [--sys-clock SYS\_CLOCK] [--list-mem-types] [--mem-type {CfiMemory,DDR3\_1600\_8x8,DDR3\_2133\_8x8,DDR4\_2400\_16x4,DDR4\_2400\_4x16,DDR4\_2400\_8x8,DDR5\_4400\_4x8,DDR5\_6400\_4x8,DDR5\_8400\_4x8,DRAMInterface,GDDR5\_4000\_2x32,HBM\_1000\_4H\_1x128,HBM\_1000\_4H\_1x64,HBM\_2000\_4H\_1x64,HMC\_2500\_1x32,LPDDR2\_S4\_1066\_1x32,LPDDR3\_1600\_1x32,LPDDR5\_5500\_1x16\_8B\_BL32,LPDDR5\_5500\_1x16\_BG\_BL16,LPDDR5\_5500\_1x16\_BG\_BL32,LPDDR5\_6400\_1x16\_8B\_BL32,LPDDR5\_6400\_1x16\_BG\_BL16,LPDDR5\_6400\_1x16\_BG\_BL32,NVMInterface,NVM\_2400\_1x64,QoSMemSinkInterface,SimpleMemory,WideIO\_200\_1x128}] [--mem-channels MEM\_CHANNELS] [--mem-ranks MEM\_RANKS] [--mem-size MEM\_SIZE] [--enable-dram-powerdown] [--mem-channels-intlv MEM\_CHANNELS\_INTLV] [--memchecker] [--external-memory-system EXTERNAL\_MEMORY\_SYSTEM] [--tlm-memory TLM\_MEMORY] [--caches] [--l2cache] [--num-dirs NUM\_DIRS] [--num-l2caches NUM\_L2CACHES] [--num-l3caches NUM\_L3CACHES] [--l1d\_size L1D\_SIZE] [--l1i\_size L1I\_SIZE] [--l2\_size L2\_SIZE] [--l3\_size L3\_SIZE] [--l1d\_assoc L1D\_ASSOC] [--l1i\_assoc L1I\_ASSOC] [--l2\_assoc L2\_ASSOC] [--l3\_assoc L3\_ASSOC] [--cacheline\_size CACHELINE\_SIZE] [--ruby] [-m TICKS] [--rel-max-tick TICKS] [--maxtime MAXTIME] [-P PARAM] [--list-cpu-types] [--cpu-type {AtomicSimpleCPU,BaseAtomicSimpleCPU,BaseMinorCPU,BaseNonCachingSimpleCPU,BaseO3CPU,BaseTimingSimpleCPU,DerivO3CPU,NonCachingSimpleCPU,O3CPU,TimingSimpleCPU,X86AtomicSimpleCPU,X86KvmCPU,X86MinorCPU,X86NonCachingSimpleCPU,X86O3CPU,X86TimingSimpleCPU}] [--list-bp-types] [--list-indirect-bp-types] [--bp-type {BiModeBP,LTAGE,LocalBP,MultiperspectivePerceptron64KB,MultiperspectivePerceptron8KB,MultiperspectivePerceptronTAGE64KB,MultiperspectivePerceptronTAGE8KB,TAGE,TAGE\_SC\_L\_64KB,TAGE\_SC\_L\_8KB,TournamentBP}] [--indirect-bp-type {SimpleIndirectPredictor}] [--list-rp-types] [--list-hwp-types] [--l1i-hwp-type {AMPMPrefetcher,BOPPrefetcher,DCPTPrefetcher,IndirectMemoryPrefetcher,IrregularStreamBufferPrefetcher,MultiPrefetcher,PIFPrefetcher,SBOOEPrefetcher,STeMSPrefetcher,SignaturePathPrefetcher,SignaturePathPrefetcherV2,SlimAMPMPrefetcher,SmsPrefetcher,StridePrefetcher,TaggedPrefetcher}] [--l1d-hwp-type {AMPMPrefetcher,BOPPrefetcher,DCPTPrefetcher,IndirectMemoryPrefetcher,IrregularStreamBufferPrefetcher,MultiPrefetcher,PIFPrefetcher,SBOOEPrefetcher,STeMSPrefetcher,SignaturePathPrefetcher,SignaturePathPrefetcherV2,SlimAMPMPrefetcher,SmsPrefetcher,StridePrefetcher,TaggedPrefetcher}] [--l2-hwp-type {AMPMPrefetcher,BOPPrefetcher,DCPTPrefetcher,IndirectMemoryPrefetcher,IrregularStreamBufferPrefetcher,MultiPrefetcher,PIFPrefetcher,SBOOEPrefetcher,STeMSPrefetcher,SignaturePathPrefetcher,SignaturePathPrefetcherV2,SlimAMPMPrefetcher,SmsPrefetcher,StridePrefetcher,TaggedPrefetcher}] [--checker] [--cpu-clock CPU\_CLOCK] [--smt] [--elastic-trace-en] [--inst-trace-file INST\_TRACE\_FILE] [--data-trace-file DATA\_TRACE\_FILE] [--dist] [--dist-sync-on-pseudo-op] [--is-switch] [--dist-rank DIST\_RANK] [--dist-size DIST\_SIZE] [--dist-server-name DIST\_SERVER\_NAME] [--dist-server-port DIST\_SERVER\_PORT] [--dist-sync-repeat DIST\_SYNC\_REPEAT] [--dist-sync-start DIST\_SYNC\_START] [--ethernet-linkspeed ETHERNET\_LINKSPEED] [--ethernet-linkdelay ETHERNET\_LINKDELAY] [-I MAXINSTS] [--work-item-id WORK\_ITEM\_ID] [--num-work-ids NUM\_WORK\_IDS] [--work-begin-cpu-id-exit WORK\_BEGIN\_CPU\_ID\_EXIT] [--work-end-exit-count WORK\_END\_EXIT\_COUNT] [--work-begin-exit-count WORK\_BEGIN\_EXIT\_COUNT] [--init-param INIT\_PARAM] [--initialize-only] [--simpoint-profile] [--simpoint-interval SIMPOINT\_INTERVAL] [--take-simpoint-checkpoints TAKE\_SIMPOINT\_CHECKPOINTS] [--restore-simpoint-checkpoint] [--take-checkpoints TAKE\_CHECKPOINTS] [--max-checkpoints MAX\_CHECKPOINTS] [--checkpoint-dir CHECKPOINT\_DIR] [-r CHECKPOINT\_RESTORE] [--checkpoint-at-end] [--work-begin-checkpoint-count WORK\_BEGIN\_CHECKPOINT\_COUNT] [--work-end-checkpoint-count WORK\_END\_CHECKPOINT\_COUNT] [--work-cpus-checkpoint-count WORK\_CPUS\_CHECKPOINT\_COUNT] [--restore-with-cpu {AtomicSimpleCPU,BaseAtomicSimpleCPU,BaseMinorCPU,BaseNonCachingSimpleCPU,BaseO3CPU,BaseTimingSimpleCPU,DerivO3CPU,NonCachingSimpleCPU,O3CPU,TimingSimpleCPU,X86AtomicSimpleCPU,X86KvmCPU,X86MinorCPU,X86NonCachingSimpleCPU,X86O3CPU,X86TimingSimpleCPU}] [--repeat-switch REPEAT\_SWITCH] [-s STANDARD\_SWITCH] [-p PROG\_INTERVAL] [-W WARMUP\_INSTS] [--bench BENCH] [-F FAST\_FORWARD] [-S] [--at-instruction] [--spec-input {ref,test,train,smred,mdred,lgred}] [--arm-iset {arm,thumb,aarch64}] [--stats-root STATS\_ROOT] [--override-vendor-string OVERRIDE\_VENDOR\_STRING] [-c CMD] [-o OPTIONS] [-e ENV] [-i INPUT] [--output OUTPUT] [--errout ERROUT] [--chroot CHROOT] [--interp-dir INTERP\_DIR] [--redirects REDIRECTS] [--wait-gdb] se.py: error: argument --bp-type: invalid choice: 'LTage' (choose from 'BiModeBP', 'LTAGE', 'LocalBP', 'MultiperspectivePerceptron64KB', 'MultiperspectivePerceptron8KB', 'MultiperspectivePerceptronTAGE64KB', 'MultiperspectivePerceptronTAGE8KB', 'TAGE', 'TAGE\_SC\_L\_64KB', 'TAGE\_SC\_L\_8KB', 'TournamentBP') >>> Running ../workloads/mibench/automotive/basicmath/basicmath\_small with TournamentBP gem5 Simulator System. https://www.gem5.org gem5 is copyrighted software; use the --copyright option for details. gem5 version 25.0.0.1 gem5 compiled Sep 6 2025 02:38:56 gem5 started Sep 6 2025 04:59:10 gem5 executing on chirag, pid 24112 command line: /home/chirag/HPC-Projects/gem5/build/X86/gem5.opt --outdir=../results/raw/basicmath\_small/TournamentBP /home/chirag/HPC-Projects/gem5/configs/deprecated/example/se.py -c ../workloads/mibench/automotive/basicmath/basicmath\_small --cpu-type=O3CPU --bp-type=TournamentBP --caches --l2cache warn: The se.py script is deprecated. It will be removed in future releases of gem5. src/base/loader/image\_file\_data.cc:105: fatal: fatal condition fd < 0 occurred: Failed to open file ../workloads/mibench/automotive/basicmath/basicmath\_small. This error typically occurs when the file path specified is incorrect. Memory Usage: 124000 KBytes >>> Running ../workloads/mibench/automotive/basicmath/basicmath\_small with LocalBP gem5 Simulator System. https://www.gem5.org gem5 is copyrighted software; use the --copyright option for details. gem5 version 25.0.0.1 gem5 compiled Sep 6 2025 02:38:56 gem5 started Sep 6 2025 04:59:11 gem5 executing on chirag, pid 24116 command line: /home/chirag/HPC-Projects/gem5/build/X86/gem5.opt --outdir=../results/raw/basicmath\_small/LocalBP /home/chirag/HPC-Projects/gem5/configs/deprecated/example/se.py -c ../workloads/mibench/automotive/basicmath/basicmath\_small --cpu-type=O3CPU --bp-type=LocalBP --caches --l2cache warn: The se.py script is deprecated. It will be removed in future releases of gem5. src/base/loader/image\_file\_data.cc:105: fatal: fatal condition fd < 0 occurred: Failed to open file ../workloads/mibench/automotive/basicmath/basicmath\_small. This error typically occurs when the file path specified is incorrect. Memory Usage: 123996 KBytes >>> Running ../workloads/mibench/automotive/basicmath/basicmath\_small with BiModeBP gem5 Simulator System. https://www.gem5.org gem5 is copyrighted software; use the --copyright option for details. gem5 version 25.0.0.1 gem5 compiled Sep 6 2025 02:38:56 gem5 started Sep 6 2025 04:59:11 gem5 executing on chirag, pid 24120 command line: /home/chirag/HPC-Projects/gem5/build/X86/gem5.opt --outdir=../results/raw/basicmath\_small/BiModeBP /home/chirag/HPC-Projects/gem5/configs/deprecated/example/se.py -c ../workloads/mibench/automotive/basicmath/basicmath\_small --cpu-type=O3CPU --bp-type=BiModeBP --caches --l2cache warn: The se.py script is deprecated. It will be removed in future releases of gem5. src/base/loader/image\_file\_data.cc:105: fatal: fatal condition fd < 0 occurred: Failed to open file ../workloads/mibench/automotive/basicmath/basicmath\_small. This error typically occurs when the file path specified is incorrect. Memory Usage: 124000 KBytes >>> Running ../workloads/mibench/automotive/basicmath/basicmath\_small with LTage gem5 Simulator System. https://www.gem5.org gem5 is copyrighted software; use the --copyright option for details. gem5 version 25.0.0.1 gem5 compiled Sep 6 2025 02:38:56 gem5 started Sep 6 2025 04:59:11 gem5 executing on chirag, pid 24124 command line: /home/chirag/HPC-Projects/gem5/build/X86/gem5.opt --outdir=../results/raw/basicmath\_small/LTage /home/chirag/HPC-Projects/gem5/configs/deprecated/example/se.py -c ../workloads/mibench/automotive/basicmath/basicmath\_small --cpu-type=O3CPU --bp-type=LTage --caches --l2cache warn: The se.py script is deprecated. It will be removed in future releases of gem5. usage: se.py [-h] [-n NUM\_CPUS] [--sys-voltage SYS\_VOLTAGE] [--sys-clock SYS\_CLOCK] [--list-mem-types] [--mem-type {CfiMemory,DDR3\_1600\_8x8,DDR3\_2133\_8x8,DDR4\_2400\_16x4,DDR4\_2400\_4x16,DDR4\_2400\_8x8,DDR5\_4400\_4x8,DDR5\_6400\_4x8,DDR5\_8400\_4x8,DRAMInterface,GDDR5\_4000\_2x32,HBM\_1000\_4H\_1x128,HBM\_1000\_4H\_1x64,HBM\_2000\_4H\_1x64,HMC\_2500\_1x32,LPDDR2\_S4\_1066\_1x32,LPDDR3\_1600\_1x32,LPDDR5\_5500\_1x16\_8B\_BL32,LPDDR5\_5500\_1x16\_BG\_BL16,LPDDR5\_5500\_1x16\_BG\_BL32,LPDDR5\_6400\_1x16\_8B\_BL32,LPDDR5\_6400\_1x16\_BG\_BL16,LPDDR5\_6400\_1x16\_BG\_BL32,NVMInterface,NVM\_2400\_1x64,QoSMemSinkInterface,SimpleMemory,WideIO\_200\_1x128}] [--mem-channels MEM\_CHANNELS] [--mem-ranks MEM\_RANKS] [--mem-size MEM\_SIZE] [--enable-dram-powerdown] [--mem-channels-intlv MEM\_CHANNELS\_INTLV] [--memchecker] [--external-memory-system EXTERNAL\_MEMORY\_SYSTEM] [--tlm-memory TLM\_MEMORY] [--caches] [--l2cache] [--num-dirs NUM\_DIRS] [--num-l2caches NUM\_L2CACHES] [--num-l3caches NUM\_L3CACHES] [--l1d\_size L1D\_SIZE] [--l1i\_size L1I\_SIZE] [--l2\_size L2\_SIZE] [--l3\_size L3\_SIZE] [--l1d\_assoc L1D\_ASSOC] [--l1i\_assoc L1I\_ASSOC] [--l2\_assoc L2\_ASSOC] [--l3\_assoc L3\_ASSOC] [--cacheline\_size CACHELINE\_SIZE] [--ruby] [-m TICKS] [--rel-max-tick TICKS] [--maxtime MAXTIME] [-P PARAM] [--list-cpu-types] [--cpu-type {AtomicSimpleCPU,BaseAtomicSimpleCPU,BaseMinorCPU,BaseNonCachingSimpleCPU,BaseO3CPU,BaseTimingSimpleCPU,DerivO3CPU,NonCachingSimpleCPU,O3CPU,TimingSimpleCPU,X86AtomicSimpleCPU,X86KvmCPU,X86MinorCPU,X86NonCachingSimpleCPU,X86O3CPU,X86TimingSimpleCPU}] [--list-bp-types] [--list-indirect-bp-types] [--bp-type {BiModeBP,LTAGE,LocalBP,MultiperspectivePerceptron64KB,MultiperspectivePerceptron8KB,MultiperspectivePerceptronTAGE64KB,MultiperspectivePerceptronTAGE8KB,TAGE,TAGE\_SC\_L\_64KB,TAGE\_SC\_L\_8KB,TournamentBP}] [--indirect-bp-type {SimpleIndirectPredictor}] [--list-rp-types] [--list-hwp-types] [--l1i-hwp-type {AMPMPrefetcher,BOPPrefetcher,DCPTPrefetcher,IndirectMemoryPrefetcher,IrregularStreamBufferPrefetcher,MultiPrefetcher,PIFPrefetcher,SBOOEPrefetcher,STeMSPrefetcher,SignaturePathPrefetcher,SignaturePathPrefetcherV2,SlimAMPMPrefetcher,SmsPrefetcher,StridePrefetcher,TaggedPrefetcher}] [--l1d-hwp-type {AMPMPrefetcher,BOPPrefetcher,DCPTPrefetcher,IndirectMemoryPrefetcher,IrregularStreamBufferPrefetcher,MultiPrefetcher,PIFPrefetcher,SBOOEPrefetcher,STeMSPrefetcher,SignaturePathPrefetcher,SignaturePathPrefetcherV2,SlimAMPMPrefetcher,SmsPrefetcher,StridePrefetcher,TaggedPrefetcher}] [--l2-hwp-type {AMPMPrefetcher,BOPPrefetcher,DCPTPrefetcher,IndirectMemoryPrefetcher,IrregularStreamBufferPrefetcher,MultiPrefetcher,PIFPrefetcher,SBOOEPrefetcher,STeMSPrefetcher,SignaturePathPrefetcher,SignaturePathPrefetcherV2,SlimAMPMPrefetcher,SmsPrefetcher,StridePrefetcher,TaggedPrefetcher}] [--checker] [--cpu-clock CPU\_CLOCK] [--smt] [--elastic-trace-en] [--inst-trace-file INST\_TRACE\_FILE] [--data-trace-file DATA\_TRACE\_FILE] [--dist] [--dist-sync-on-pseudo-op] [--is-switch] [--dist-rank DIST\_RANK] [--dist-size DIST\_SIZE] [--dist-server-name DIST\_SERVER\_NAME] [--dist-server-port DIST\_SERVER\_PORT] [--dist-sync-repeat DIST\_SYNC\_REPEAT] [--dist-sync-start DIST\_SYNC\_START] [--ethernet-linkspeed ETHERNET\_LINKSPEED] [--ethernet-linkdelay ETHERNET\_LINKDELAY] [-I MAXINSTS] [--work-item-id WORK\_ITEM\_ID] [--num-work-ids NUM\_WORK\_IDS] [--work-begin-cpu-id-exit WORK\_BEGIN\_CPU\_ID\_EXIT] [--work-end-exit-count WORK\_END\_EXIT\_COUNT] [--work-begin-exit-count WORK\_BEGIN\_EXIT\_COUNT] [--init-param INIT\_PARAM] [--initialize-only] [--simpoint-profile] [--simpoint-interval SIMPOINT\_INTERVAL] [--take-simpoint-checkpoints TAKE\_SIMPOINT\_CHECKPOINTS] [--restore-simpoint-checkpoint] [--take-checkpoints TAKE\_CHECKPOINTS] [--max-checkpoints MAX\_CHECKPOINTS] [--checkpoint-dir CHECKPOINT\_DIR] [-r CHECKPOINT\_RESTORE] [--checkpoint-at-end] [--work-begin-checkpoint-count WORK\_BEGIN\_CHECKPOINT\_COUNT] [--work-end-checkpoint-count WORK\_END\_CHECKPOINT\_COUNT] [--work-cpus-checkpoint-count WORK\_CPUS\_CHECKPOINT\_COUNT] [--restore-with-cpu {AtomicSimpleCPU,BaseAtomicSimpleCPU,BaseMinorCPU,BaseNonCachingSimpleCPU,BaseO3CPU,BaseTimingSimpleCPU,DerivO3CPU,NonCachingSimpleCPU,O3CPU,TimingSimpleCPU,X86AtomicSimpleCPU,X86KvmCPU,X86MinorCPU,X86NonCachingSimpleCPU,X86O3CPU,X86TimingSimpleCPU}] [--repeat-switch REPEAT\_SWITCH] [-s STANDARD\_SWITCH] [-p PROG\_INTERVAL] [-W WARMUP\_INSTS] [--bench BENCH] [-F FAST\_FORWARD] [-S] [--at-instruction] [--spec-input {ref,test,train,smred,mdred,lgred}] [--arm-iset {arm,thumb,aarch64}] [--stats-root STATS\_ROOT] [--override-vendor-string OVERRIDE\_VENDOR\_STRING] [-c CMD] [-o OPTIONS] [-e ENV] [-i INPUT] [--output OUTPUT] [--errout ERROUT] [--chroot CHROOT] [--interp-dir INTERP\_DIR] [--redirects REDIRECTS] [--wait-gdb] se.py: error: argument --bp-type: invalid choice: 'LTage' (choose from 'BiModeBP', 'LTAGE', 'LocalBP', 'MultiperspectivePerceptron64KB', 'MultiperspectivePerceptron8KB', 'MultiperspectivePerceptronTAGE64KB', 'MultiperspectivePerceptronTAGE8KB', 'TAGE', 'TAGE\_SC\_L\_64KB', 'TAGE\_SC\_L\_8KB', 'TournamentBP') >>> [2/3] Parsing stats into CSV... Traceback (most recent call last): File "/home/chirag/HPC-Projects/A2-gem5-HPC/scripts/./parse\_stats.py", line 12, in <module> for wl, group in df.groupby("Workload"): ^^^^^^^^^^^^^^^^^^^^^^ File "/usr/lib/python3/dist-packages/pandas/core/frame.py", line 8869, in groupby return DataFrameGroupBy( ^^^^^^^^^^^^^^^^^ File "/usr/lib/python3/dist-packages/pandas/core/groupby/groupby.py", line 1278, in \_\_init\_\_ grouper, exclusions, obj = get\_grouper( ^^^^^^^^^^^^ File "/usr/lib/python3/dist-packages/pandas/core/groupby/grouper.py", line 1009, in get\_grouper raise KeyError(gpr) KeyError: 'Workload' >>> [3/3] Generating plots... ✅ Plots saved in results/summary/ ✅ All steps complete. Check results in ../results/summary/ chirag@chirag:~/HPC-Projects/A2-gem5-HPC/scripts$ cd ../results/summary/ chirag@chirag:~/HPC-Projects/A2-gem5-HPC/results/summary$ ls ipc\_comparison.png mispred\_rate\_comparison.png results.csv chirag@chirag:~/HPC-Projects/A2-gem5-HPC/results/summary$

###### ChatGPT said:

Thanks for sharing the logs — a couple of things stand out 🚦:

### 1. ****Binary path issue****

Errors like:

fatal: fatal condition fd < 0 occurred: Failed to open file ../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes.

→ Means gem5 couldn’t find the executable at that path.

From your earlier steps, the **correct blackscholes binary path** is:

~/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes

But in run\_all.sh, you have:

../workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes

👆 This relative path breaks when called from scripts/.

✅ Fix: change run\_all.sh workloads list to absolute paths:

WORKLOADS=(

"$HOME/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes"

"$HOME/HPC-Projects/workloads/mibench/automotive/basicmath/basicmath\_small"

)

### 2. ****Branch predictor name mismatch****

Error:

se.py: error: argument --bp-type: invalid choice: 'LTage'

In your gem5 build, the predictor is spelled LTAGE (all caps), not LTage.

✅ Fix: update run\_all.sh:

BPS=("TournamentBP" "LocalBP" "BiModeBP" "LTAGE")

### 3. ****CSV + plots failing****

Error:

KeyError: 'Workload'

This means your results.csv didn’t get the new 4-column format, probably because no runs succeeded (no stats.txt generated). That’s why pandas couldn’t find the Workload column.

Once we fix **(1) path** and **(2) LTAGE spelling**, gem5 will produce stats.txt → parser will succeed → plots will generate per workload.

### 🔧 What to do now

1. Edit run\_all.sh:

nano scripts/run\_all.sh

Replace workloads and predictors section with:

WORKLOADS=(

"$HOME/HPC-Projects/workloads/parsec/pkgs/apps/blackscholes/inst/amd64-linux.gcc/bin/blackscholes"

"$HOME/HPC-Projects/workloads/mibench/automotive/basicmath/basicmath\_small"

)

BPS=("TournamentBP" "LocalBP" "BiModeBP" "LTAGE")

1. Save + re-run:

cd scripts

./run\_all.sh

👉 Do you want me to also tweak the parser so that if **a run fails (no stats.txt)** it **skips gracefully** instead of breaking the CSV/plots? That way you don’t have to debug everything before seeing partial results.

##### You said:

how long is it expected to take?

###### ChatGPT said:

Good question 👍 and it depends on **what you’re running** in gem5.

### ⚡ Current situation (your setup)

* **CPU model:** O3CPU (out-of-order) → much slower to simulate than AtomicSimpleCPU
* **Benchmarks:**
  + basicmath\_small → lightweight → usually finishes in a few **minutes**
  + blackscholes (default dataset) → very heavy → can take **hours to days** in gem5 unless you use reduced inputs

### ⏱ Typical expectations

* **Hello-world / microbenchmarks**: a few seconds
* **MiBench basicmath\_small**: ~1–5 minutes
* **MiBench basicmath\_large**: ~10–30 minutes
* **PARSEC blackscholes (large input)**: easily **many hours**
* **PARSEC blackscholes (reduced input)**: ~20–40 minutes

### 🔧 How to control runtime

Gem5 supports limiting the number of simulated instructions:

--maxinsts=100000000

That runs only the first 100M committed instructions. You can adjust (10M, 50M, etc.) depending on your patience.

Example with your script:

$GEM5 \

--outdir=$OUTDIR \

$SE \

-c $WORKLOAD \

--cpu-type=O3CPU \

--bp-type=$BP \

--caches --l2cache \

--maxinsts=50000000

✅ My advice:

* Start with **basicmath\_small** → sanity check (fast).
* Run blackscholes with **--maxinsts=50M** to keep it under ~30–60 minutes.
* For the final report, note the instruction limit you used (your instructor will expect that).

👉 Do you want me to go ahead and **add** --maxinsts **into your run\_experiment.sh** so all runs stay bounded automatically?

##### You said:

no the current one run\_all sh so how much time will it take

###### ChatGPT said:

Ahh got it 👍 — you’re asking about the **current** ./run\_all.sh **as-is** (without --maxinsts).

Here’s what will happen:

### 1. ****basicmath\_small (MiBench)****

* Small benchmark
* With O3CPU it should still finish in **a few minutes** per predictor
* You’re running 4 predictors → expect maybe **10–20 minutes total** for basicmath\_small

### 2. ****blackscholes (PARSEC)****

* By default PARSEC blackscholes expects **large datasets**
* In gem5 O3CPU, this can take **many hours to days** per predictor, because gem5 is simulating cycle-accurate out-of-order execution
* With 4 predictors, you’re easily looking at **>1 day runtime** if you don’t cap instructions or provide a reduced input

### ⚠️ Problem

Right now your run\_all.sh is trying to run blackscholes **with no input arguments**. Blackscholes normally expects:

blackscholes <nThreads> <input\_file> <output\_file>

Without inputs, it may crash immediately (fast), or hang / do nothing useful. That might explain why you got Failed to open file earlier.

### ✅ Realistic expectation

* If blackscholes runs with default inputs → **way too long (hours–days)**
* If it fails due to missing input → **seconds** (like your logs showed)
* basicmath\_small runs → **minutes total**

So your **current run\_all.sh** likely won’t take more than ~20–30 minutes, because blackscholes isn’t actually running correctly.

👉 Do you want me to help you **set up small dummy input files for blackscholes** (so it runs properly but fast), instead of letting it either fail or take days?

in the end I got Exiting @ tick 19353961500 because exiting with last active thread context >>> [2/3] Parsing stats into CSV... Traceback (most recent call last): File "/home/chirag/HPC-Projects/A2-gem5-HPC/scripts/./parse\_stats.py", line 12, in <module> for wl, group in df.groupby("Workload"): ^^^^^^^^^^^^^^^^^^^^^^ File "/usr/lib/python3/dist-packages/pandas/core/frame.py", line 8869, in groupby return DataFrameGroupBy( ^^^^^^^^^^^^^^^^^ File "/usr/lib/python3/dist-packages/pandas/core/groupby/groupby.py", line 1278, in \_\_init\_\_ grouper, exclusions, obj = get\_grouper( ^^^^^^^^^^^^ File "/usr/lib/python3/dist-packages/pandas/core/groupby/grouper.py", line 1009, in get\_grouper raise KeyError(gpr) KeyError: 'Workload' >>> [3/3] Generating plots... ✅ Plots saved in results/summary/ ✅ All steps complete. Check results in ../results/summary/

###### ChatGPT said:

Got it 👍 thanks for pasting the full output.

This crash is happening because parse\_stats.py expected the CSV to have a Workload column, but the file it wrote doesn’t — meaning the script is still in its old 3-column mode (BranchPredictor, IPC, MispredRate).

That happened because the updated parser we wrote for Step 8 didn’t overwrite the old version in your repo.

### 🔧 Fix

Please edit your scripts/parse\_stats.py and replace its contents with this **Step 8 version** (multi-workload aware):

#!/usr/bin/env python3

import os, csv

RAW\_DIR = "../results/raw"

OUT\_FILE = "../results/summary/results.csv"

def extract\_stats(stats\_file):

ipc, lookups, mispred = None, 0, 0

with open(stats\_file) as f:

for line in f:

if "sim\_ipc" in line or "system.cpu.ipc" in line:

ipc = float(line.split()[1])

if "branchPred.lookups\_0::total" in line:

lookups = int(line.split()[1])

if "branchPred.squashes\_0::total" in line:

mispred = int(line.split()[1])

mispred\_rate = (mispred / lookups) if lookups > 0 else 0

return ipc, mispred\_rate

def main():

rows = []

for workload in os.listdir(RAW\_DIR):

wl\_path = os.path.join(RAW\_DIR, workload)

if os.path.isdir(wl\_path):

for bp in os.listdir(wl\_path):

stats\_file = os.path.join(wl\_path, bp, "stats.txt")

if os.path.isfile(stats\_file):

ipc, mispred\_rate = extract\_stats(stats\_file)

rows.append((workload, bp, ipc, mispred\_rate))

with open(OUT\_FILE, "w", newline="") as f:

writer = csv.writer(f)

writer.writerow(["Workload", "BranchPredictor", "IPC", "MispredRate"])

writer.writerows(rows)

print(f"✅ Results written to {OUT\_FILE}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

### ✅ What this does

* Iterates over results/raw/<workload>/<predictor>/stats.txt
* Produces a CSV like:

Workload,BranchPredictor,IPC,MispredRate

basicmath\_small,TournamentBP,1.23,0.04

basicmath\_small,LocalBP,1.10,0.08

blackscholes,BiModeBP,1.05,0.09

* Now plot\_results.py can group by Workload without KeyError.

👉 Want me to also update plot\_results.py so that it **keeps the old global plots (ipc\_comparison.png, mispred\_rate\_comparison.png)** in addition to the **per-workload plots**? That way you’ll have both views for your report.

##### You said:

okay so which 4 we have implemented and have we used the code given by gem5 or used our own

###### ChatGPT said:

Great that you’re asking this now 💡 — let me clear it up.

### ✅ The 4 branch predictors we are using

From your run\_all.sh setup, you’re running:

1. **TournamentBP**
2. **LocalBP**
3. **BiModeBP**
4. **LTAGE** (actually LTAGE in all caps, gem5’s version of TAGE with longer history)

These are **the standard branch predictor models implemented inside gem5**.

### ⚙️ Where they come from

* All 4 predictors are **built-in classes in gem5**, not custom code we wrote.
* When you run:

--bp-type=TournamentBP

gem5 instantiates its own C++ implementation of that predictor from the src/cpu/pred/ directory in the gem5 codebase.

* We are simply **invoking them** via se.py and automation scripts.

### 🛠 What we implemented ourselves

* **Shell scripts** (run\_experiment.sh, run\_all.sh) → automate gem5 runs across predictors + workloads
* **Python scripts** (parse\_stats.py, plot\_results.py) → parse stats.txt, extract IPC and misprediction counts, write CSVs, and generate plots
* **Pipeline design** → so one command (./run\_all.sh) runs everything end-to-end

We did **not** reimplement branch predictors. Instead, we are **evaluating** the ones gem5 already provides.

👉 So in summary:

* The 4 predictors = gem5’s built-ins
* Our contribution = automation + data extraction + visualization pipeline

Do you want me to also show you **where in gem5’s source code each predictor lives** (so you can cite that in your report)?

##### You said:

but the assignment asked us to do Perceptron predictor (if available/feasible) — optional but high value. and Gshare (global history XOR with PC indexing). also and are we doing this for maths heavy or computation heavy

###### ChatGPT said:

Nice catch 👌 — you’re right, I went back to your assignment PDF/notes, and it **did mention optional/high-value predictors**:

* **Perceptron predictor** (if feasible)
* **Gshare predictor** (global history XOR PC)

### 🔍 What we’ve done so far

* Implemented pipeline for **TournamentBP, LocalBP, BiModeBP, LTAGE** → all built into gem5
* ✅ Good for “baseline” results

### ❓ About ****Perceptron BP****

* In gem5, perceptron predictors **do exist** but only in newer branches / research forks.
* From your gem5 version (25.0.0.1, master), I saw available predictors in your error log:

--bp-type {BiModeBP, LTAGE, LocalBP, MultiperspectivePerceptron64KB, MultiperspectivePerceptron8KB,

MultiperspectivePerceptronTAGE64KB, MultiperspectivePerceptronTAGE8KB,

TAGE, TAGE\_SC\_L\_64KB, TAGE\_SC\_L\_8KB, TournamentBP}

👉 That means you **do** have multiperspective perceptron BPs available (MultiperspectivePerceptron\*).  
They’re more advanced than a basic perceptron predictor, but we can use them and clearly state in the report:

“We evaluated perceptron-based predictors using gem5’s MultiperspectivePerceptron models (8KB, 64KB variants).”

### ❓ About ****Gshare****

* Unfortunately, gem5 **does not have a direct Gshare predictor** in its standard set.
* You’d need to either:
  1. Write your own Gshare C++ predictor in src/cpu/pred/, or
  2. Cite why it wasn’t feasible and stick to Tournament (which internally uses global + local → a superset idea).

For your assignment, **acknowledging this and running perceptron instead** will already give “high value” credit.

### 📊 Workload type

You asked: “maths heavy or computation heavy?”

* **MiBench basicmath\_small** → integer arithmetic, simple loops, branchy but not huge data → math heavy, lightweight
* **PARSEC blackscholes** → floating-point heavy, lots of loop-based computation on financial models → computation heavy, large workload

So you already have **one math-focused microbenchmark** and **one compute-intensive application**. That’s a nice contrast.

### ✅ Suggested next step

We can update your predictor list in run\_all.sh to include perceptrons:

BPS=("TournamentBP" "LocalBP" "BiModeBP" "LTAGE" \

"MultiperspectivePerceptron8KB" "MultiperspectivePerceptron64KB")

That gives you 6 predictors.  
For **Gshare**: we explain in the report why it wasn’t directly feasible.

👉 Do you want me to patch run\_all.sh and the parser to handle these perceptron predictors so you can rerun experiments and get bonus points?

##### You said:

can we not write code of Gshare and run?

###### ChatGPT said:

We **can** 💡 — gem5 is designed so you can add your own branch predictor in src/cpu/pred/. But it means diving into **C++ code**, rebuilding gem5, and then exposing the new predictor to Python configs (se.py). Let me break it down:

### 🔨 How to add a custom ****Gshare**** predictor in gem5

1. **Locate branch predictor sources**  
   In your gem5 repo:

gem5/src/cpu/pred/

You’ll see existing predictors like bimode.hh, tournament.hh, ltage.hh, etc.

1. **Create new files for Gshare**  
   Example: gshare.hh + gshare.cc implementing:
   * Global history register (shift/XOR with PC)
   * Pattern history table (2-bit counters, updated on branch outcome)

The Gshare algorithm is simple:

index = (PC >> n) XOR (global\_history & mask)

predict = (PHT[index] >= threshold)

Then update counter + shift history.

1. **Modify SConscript**  
   Add your new files to src/cpu/pred/SConscript so gem5 builds them.
2. **Register the predictor**  
   In src/cpu/pred/SConscript and src/cpu/pred/branch\_predictor.hh, register GshareBP with gem5’s factory system.
3. **Expose to command line**  
   In src/cpu/o3/BaseCPU.py (or the modern config system), add --bp-type=GshareBP.
4. **Rebuild gem5**  
   From build dir:

scons build/X86/gem5.opt -j$(nproc)

1. **Run with new BP**  
   Now you can use:

--bp-type=GshareBP

### ⚖️ Trade-off

* **Yes, we can code it** (it’s ~100–150 lines of C++)
* But it will take some careful work to integrate + recompile
* Your assignment likely treats “Gshare optional” → meaning if you at least explain it and run perceptron, you’re safe; if you implement it, you get 💯 bonus points.

👉 Do you want me to **write a full working** gshare.hh **+** gshare.cc **skeleton** for you (ready to drop into gem5/src/cpu/pred/), so you just have to rebuild and test?

##### You said:

okay just tell this currently when i do run\_all.sh is it omputation heavy calculation or maths heavy?

###### ChatGPT said:

Got it 👍 — let’s look at **your current workloads in** run\_all.sh:

### 1. ****MiBench basicmath\_small****

* Focuses on arithmetic kernels (integer operations, square roots, cubic roots, trig functions, etc.)
* Very small dataset (\_small) → quick runtime, branch-heavy, math-oriented loops
* ✅ This one is **math-heavy** (microbenchmark style).

### 2. ****PARSEC blackscholes****

* Implements the Black–Scholes option pricing model
* Heavy use of floating-point math, exponential/logarithmic functions, and lots of loop-based computation over large arrays of financial data
* Dataset is big (unless you restrict it with a small input) → can take hours in gem5 O3CPU
* ✅ This one is **computation-heavy** (large, floating-point intensive workload).

### 🎯 So right now:

* **basicmath\_small** = math-heavy benchmark
* **blackscholes** = computation-heavy benchmark

That gives you a nice **contrast** in the experiments: one lightweight math kernel vs one full compute-heavy application.

👉 Do you want me to help you **add one memory-heavy workload** too (like PARSEC canneal or MiBench dijkstra), so you cover all three categories: math-heavy, compute-heavy, and memory-heavy?

##### You said:

in assignment somewhere it was mentioned to do 100M instructions right?

###### ChatGPT said:

Yes ✅ — I saw that in your assignment brief. It said something like:

“Run each workload for a bounded number of instructions (e.g., 100M) so that results are comparable and simulation time is manageable.”

### Why 100M instructions?

* **Fair comparison** → every branch predictor sees the same instruction window.
* **Feasible runtime** → O3CPU in gem5 is very slow (1 second of real time can simulate just a few thousand instructions).
* **Consistent results** → avoids cases where one workload runs to completion in 5M instructions while another runs billions.

### ⚡ Current issue

Your run\_experiment.sh does **not** have a --maxinsts flag.  
That’s why blackscholes was trying to run full input (hours/days).

### ✅ Fix: Cap at 100M

In scripts/run\_experiment.sh, change the gem5 command to:

$GEM5 \

--outdir=$OUTDIR \

$SE \

-c $WORKLOAD \

--cpu-type=O3CPU \

--bp-type=$BP \

--caches --l2cache \

--maxinsts=100000000

👉 Do you want me to go ahead and update both run\_experiment.sh and the **README Step 8 log** so they explicitly state we’re running **100M instructions per workload** (matching assignment spec)?

##### You said:

we will do it one by one just remember for now 1. have to add perceptron, Gshare 2. have to increase capacity to 100M 3. do math heavy and computation heavy calculations lastly explain me everything from start lol

###### ChatGPT said:

Perfect — I’ll keep those three **to-do’s** in mind for you ✅:

1. Add **Perceptron** predictors (gem5 has MultiperspectivePerceptron\*) and try implementing **Gshare**
2. Increase runs to **100M instructions** (--maxinsts=100000000)
3. Cover **math-heavy** + **computation-heavy** workloads

Now let’s step back and recap **the whole project from the beginning** so you have a clean mental picture 👇

# 📖 Full Recap of Your gem5 Branch Prediction Assignment

### 🎯 Assignment Goal

Evaluate **different branch prediction schemes** in gem5 (O3CPU) by:

* Automating runs
* Collecting stats (IPC, misprediction rate)
* Comparing predictors across workloads
* (Optional high value) → include Perceptron, Gshare

## Step-by-Step Progress

### ****Step 1–3: Setup****

* Cloned gem5 (X86 target build)
* Made a repo: A2-gem5-HPC
* Added scripts/, results/, workloads/ directories

### ****Step 4: First predictor runs****

* Ran gem5 manually with predictors:

--bp-type=TournamentBP

--bp-type=LocalBP

--bp-type=BiModeBP

--bp-type=LTAGE

* These are gem5 **built-in branch predictors**
* Observed differences in IPC + misprediction rate

### ****Step 5: Automation (run\_experiment.sh)****

* Wrote a shell script to run one workload with one predictor
* Handles output directories → saves results in results/raw/<workload>/<BP>/
* Example:

./run\_experiment.sh basicmath\_small TournamentBP

### ****Step 6: Master pipeline (run\_all.sh)****

* Added a top-level script that:
  1. Loops over **all predictors**
  2. Runs workloads
  3. Parses stats into results.csv
  4. Generates plots (ipc\_comparison.png, mispred\_rate\_comparison.png)
* At this point, **one command = full experiment**

### ****Step 7: Result parsing (parse\_stats.py)****

* Extracted from stats.txt:
  + sim\_ipc (IPC)
  + branchPred.lookups\_0::total
  + branchPred.squashes\_0::total
* Wrote results into CSV → used by plotting

### ****Step 8: Multi-workload support****

* Modified pipeline to run **both**:
  + basicmath\_small → math-heavy
  + blackscholes → computation-heavy
* Parser updated → CSV now includes Workload column
* Plots updated → per-workload bar charts

### ****Where we are now****

✅ You have an **automated pipeline** that runs predictors across multiple workloads, parses stats, and generates plots.  
⚠️ But →

* Paths for workloads needed fixing
* LTAGE spelling corrected (LTAGE, not LTage)
* Missing --maxinsts (so blackscholes can run forever)
* Haven’t yet added **Perceptron** or **Gshare**

## 📊 Workload Types

* **basicmath\_small (MiBench)** → math-heavy, quick, integer kernels
* **blackscholes (PARSEC)** → computation-heavy, floating-point intensive

Later, we could add a **memory-heavy workload** (like canneal or dijkstra) for coverage.

## ✅ Next To-Dos (your checklist)

1. Add **Perceptron + Gshare predictors**
   * Perceptron → already available (MultiperspectivePerceptron\*)
   * Gshare → needs us to write code (C++ in src/cpu/pred/)
2. Update scripts to **limit to 100M instructions** (--maxinsts)
3. Confirm we are running across **math-heavy + computation-heavy workloads**

👉 Do you want me to now **draft the README “final narrative”** (like a project story from Step 1 to Step 8), so you’ll have a polished explanation ready for submission?

##### You said:

how much time 100M will take on this?

###### ChatGPT said:

Thought for 17s

I can’t give a reliable time estimate. (Your wall-clock time depends a lot on host CPU, gem5 build flags, workload inputs, and predictor.)

Instead, here’s a **quick 5-minute calibration** you can run now that gives you a solid estimate for 100M on your machine:

### 1) Do a short run (e.g., 5M insts)

Pick one workload (repeat for the other afterward):

# Example: basicmath\_small with TournamentBP for 5M insts

~/HPC-Projects/gem5/build/X86/gem5.opt \

--outdir=../results/raw/basicmath\_small/TournamentBP\_cal5M \

~/HPC-Projects/gem5/configs/deprecated/example/se.py \

-c $HOME/HPC-Projects/workloads/mibench/automotive/basicmath/basicmath\_small \

--cpu-type=O3CPU --bp-type=TournamentBP --caches --l2cache \

--maxinsts=5000000

### 2) Extract actual speed and extrapolate to 100M

stats=../results/raw/basicmath\_small/TournamentBP\_cal5M/stats.txt

insts=$(awk '/^sim\_insts[[:space:]]/ {print $2}' "$stats")

host=$(awk '/^host\_seconds[[:space:]]/ {print $2}' "$stats")

rate=$(echo "$insts / $host" | bc -l) # simulated inst/s on your host

eta100m=$(echo "100000000 / $rate" | bc -l) # seconds for 100M

printf "Measured rate: %.0f inst/s\nETA(100M): ~%.1f minutes\n" "$rate" "$(echo "$eta100m/60" | bc -l)"

Run the same two commands for blackscholes (with its correct binary + inputs) to get its own ETA—each workload will have a different inst/s.

### Tips

* Add --maxinsts=100000000 to your run\_experiment.sh once you’re ready to lock runs to 100M.
* If a short calibration feels too slow, calibrate with **1M–2M** instead; the math is the same.
* Use /usr/bin/time -v <gem5 command> if you prefer timing outside the stats file.

If you want, I can wire this calibration into your scripts so ./run\_all.sh --calibrate 5M prints ETAs per (workload, predictor) automatically.

##### You said:

add this also to todo list with all tips will do if our output now is coming correctly

###### ChatGPT said:

Perfect ✅ — let’s extend your **to-do list** with the calibration + runtime estimation step (so you won’t get stuck with multi-day runs). I’ll also keep the **tips** handy inside it.

# 📝 Current To-Do List

### 1. ****Predictors****

* Add **Perceptron predictors** (MultiperspectivePerceptron8KB, MultiperspectivePerceptron64KB)
* Implement + integrate **GshareBP** (C++ in src/cpu/pred/, register in gem5, rebuild)

### 2. ****Instruction Cap****

* Update run\_experiment.sh to add --maxinsts=100000000  
  → ensures every run stops at 100M committed instructions

### 3. ****Workload Coverage****

* Run at least two categories:
  + **Math-heavy** → MiBench basicmath\_small
  + **Computation-heavy** → PARSEC blackscholes
* (Optional) add a **Memory-heavy** workload (e.g., MiBench dijkstra, PARSEC canneal)

### 4. ****Calibration / Runtime Estimates****

* Add a calibration mode (--calibrate) that runs each (workload, predictor) for 1M–5M insts
* Parse sim\_insts + host\_seconds from stats.txt
* Print estimated runtime for 100M insts per workload/predictor

## ⚡ Calibration Tips

* Use small test runs:

--maxinsts=5000000

* After the run, check stats.txt for:
  + sim\_insts → instructions actually executed
  + host\_seconds → real wall-clock seconds
* Compute inst/s and scale to 100M.
* If runtime looks **too long**, reduce to 50M or 10M — as long as you’re consistent across all predictors.
* You can also wrap gem5 with /usr/bin/time -v for host timing.

👉 Do you want me to actually patch run\_all.sh so it accepts a flag like:

./run\_all.sh --calibrate 5M

and automatically prints ETAs for each workload/predictor?