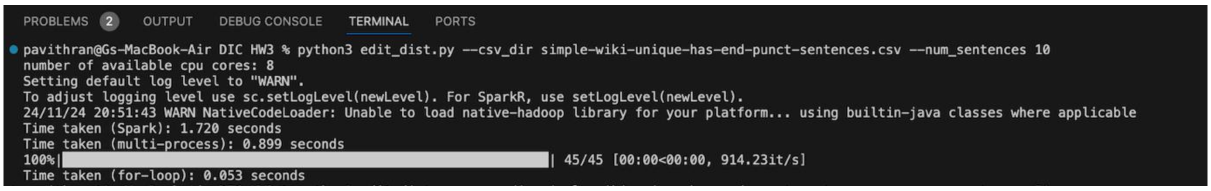
HW 3 Report

Task 1.1

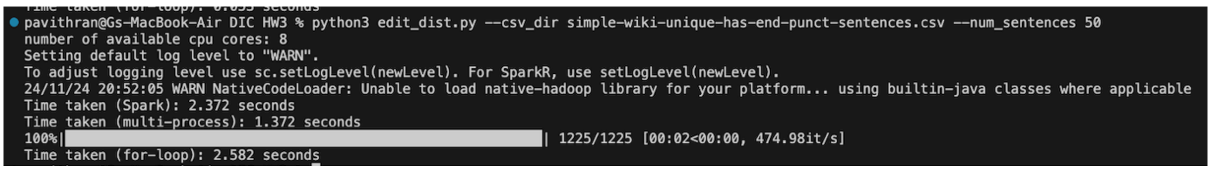
Report time cost of three implementation with different numbers of sentences: [10, 50, 100, 250, 500, 1000]

1) 10



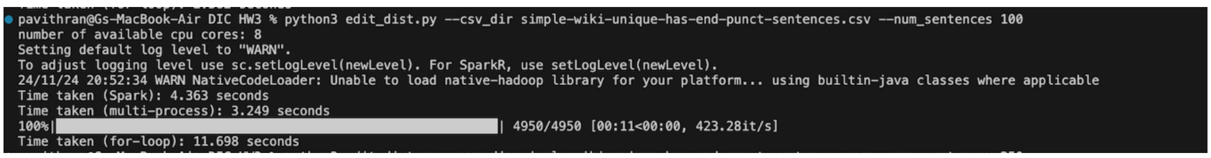
```
pavithran@Gs-MacBook-Air DIC HW3 % python3 edit_dist.py --csv_dir simple-wiki-unique-has-end-punct-sentences.csv --num_sentences 10
number of available cpu cores: 8
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/11/24 20:51:43 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Time taken (Spark): 1.720 seconds
Time taken (multi-process): 0.899 seconds
100%|████████████████████████████████████████| 45/45 [00:00<00:00, 914.23it/s]
Time taken (for-loop): 0.053 seconds
```

2) 50



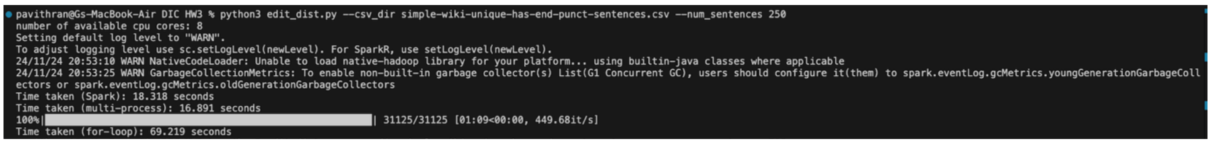
```
pavithran@Gs-MacBook-Air DIC HW3 % python3 edit_dist.py --csv_dir simple-wiki-unique-has-end-punct-sentences.csv --num_sentences 50
number of available cpu cores: 8
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/11/24 20:52:05 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Time taken (Spark): 2.372 seconds
Time taken (multi-process): 1.372 seconds
100%|████████████████████████████████████████| 1225/1225 [00:02<00:00, 474.98it/s]
Time taken (for-loop): 2.582 seconds
```

3) 100



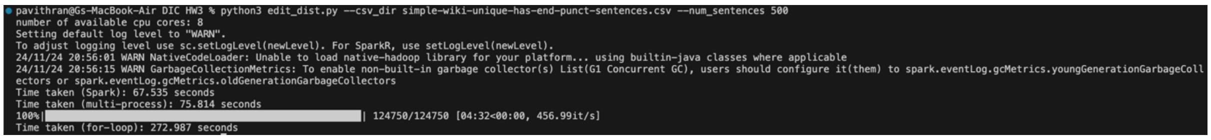
```
pavithran@Gs-MacBook-Air DIC HW3 % python3 edit_dist.py --csv_dir simple-wiki-unique-has-end-punct-sentences.csv --num_sentences 100
number of available cpu cores: 8
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/11/24 20:52:34 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Time taken (Spark): 4.363 seconds
Time taken (multi-process): 3.249 seconds
100%|████████████████████████████████████████| 4950/4950 [00:11<00:00, 423.28it/s]
Time taken (for-loop): 11.698 seconds
```

4) 250



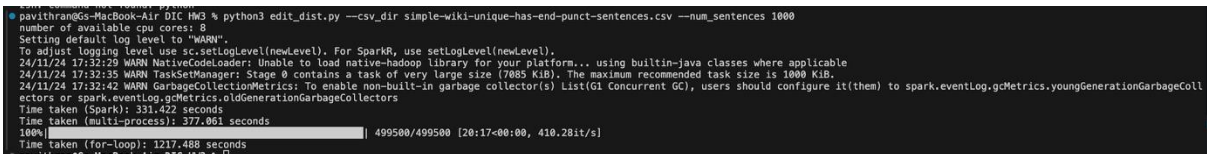
```
pavithran@Gs-MacBook-Air DIC HW3 % python3 edit_dist.py --csv_dir simple-wiki-unique-has-end-punct-sentences.csv --num_sentences 250
number of available cpu cores: 8
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/11/24 20:53:10 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
24/11/24 20:53:25 WARN GarbageCollectionMetrics: To enable non-built-in garbage collector(s) List(G1 Concurrent GC), users should configure it(them) to spark.eventLog.gcMetrics.youngGenerationGarbageColl
ectors or spark.eventLog.gcMetrics.oldGenerationGarbageCollectors
Time taken (Spark): 18.318 seconds
Time taken (multi-process): 16.891 seconds
100%|████████████████████████████████████████| 31125/31125 [01:09<00:00, 449.68it/s]
Time taken (for-loop): 69.219 seconds
```

5) 500



```
pavithran@Gs-MacBook-Air DIC HW3 % python3 edit_dist.py --csv_dir simple-wiki-unique-has-end-punct-sentences.csv --num_sentences 500
number of available cpu cores: 8
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/11/24 20:56:01 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
24/11/24 20:56:15 WARN GarbageCollectionMetrics: To enable non-built-in garbage collector(s) List(G1 Concurrent GC), users should configure it(them) to spark.eventLog.gcMetrics.youngGenerationGarbageColl
ectors or spark.eventLog.gcMetrics.oldGenerationGarbageCollectors
Time taken (Spark): 67.535 seconds
Time taken (multi-process): 75.814 seconds
100%|████████████████████████████████████████| 124750/124750 [04:32<00:00, 456.99it/s]
Time taken (for-loop): 272.987 seconds
```

6) 1000



```
pavithran@Gs-MacBook-Air DIC HW3 % python3 edit_dist.py --csv_dir simple-wiki-unique-has-end-punct-sentences.csv --num_sentences 1000
number of available cpu cores: 8
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/11/24 17:32:29 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
24/11/24 17:32:35 WARN TaskSetManager: Stage 0 contains a task of very large size (7085 KiB). The maximum recommended task size is 1000 KiB.
24/11/24 17:32:42 WARN GarbageCollectionMetrics: To enable non-built-in garbage collector(s) List(G1 Concurrent GC), users should configure it(them) to spark.eventLog.gcMetrics.youngGenerationGarbageColl
ectors or spark.eventLog.gcMetrics.oldGenerationGarbageCollectors
Time taken (Spark): 331.422 seconds
Time taken (multi-process): 377.061 seconds
100%|████████████████████████████████████████| 499500/499500 [20:17<00:00, 410.28it/s]
Time taken (for-loop): 1217.488 seconds
```

Number of sentences	Spark	Multi-Process	For-loop
10	1.72	0.89	0.053
50	2.37	1.372	2.582
100	4.36	3.24	11.69
250	18.31	16.89	69.21
500	67.53	75.81	272.98
1000	331.42	377.06	1217.48

For-loop is the slowest and least scalable method, with execution time rising significantly as input size increases.

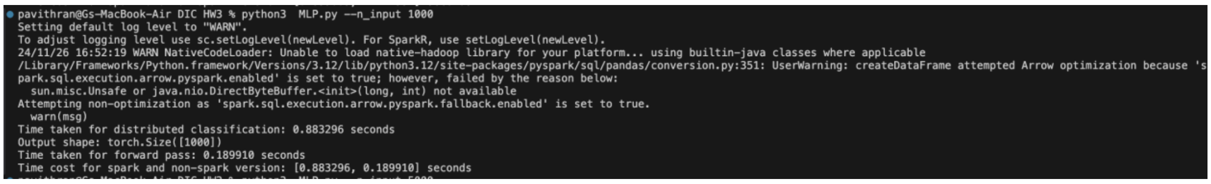
Multi-Process performs better than the For-loop but struggles with larger datasets, especially beyond 500 sentences.

Spark is the fastest and most scalable approach, maintaining superior performance even as input size grows

Task 1.2

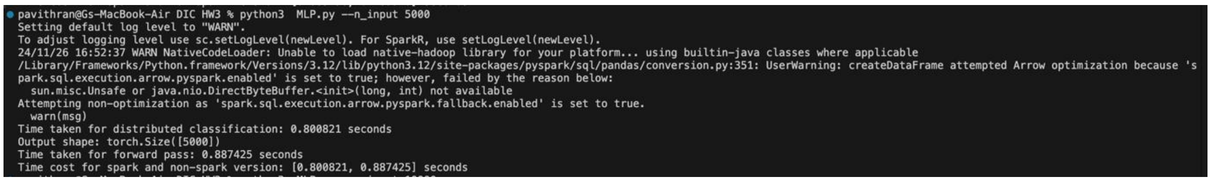
Task 1.2, test and report the time costs with values for n_input set to [1000, 5000, 10000, 50000, 100000], keeping hidden_dim and hidden_layer as default

1) 1000



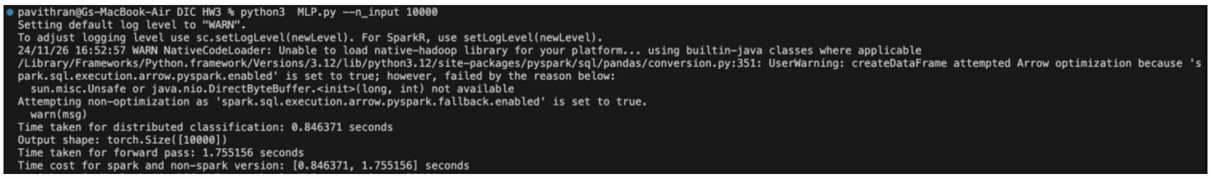
```
pavithran@Gs-MacBook-Air DIC HW3 % python3  MLP.py --n_input 1000
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/11/26 16:52:19 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/pyspark/sql/pandas/conversion.py:351: UserWarning: createDataFrame attempted Arrow optimization because 's
park.sql.execution.arrow.pyspark.enabled' is set to true; however, failed by the reason below:
  sun.misc.Unsafe or java.nio.DirectByteBuffer.<init>(long, int) not available
Attempting non-optimization as 'spark.sql.execution.arrow.pyspark.fallback.enabled' is set to true.
  warn(msg)
Time taken for distributed classification: 0.883296 seconds
Output shape: torch.Size([1000])
Time taken for forward pass: 0.189910 seconds
Time cost for spark and non-spark version: [0.883296, 0.189910] seconds
```

2) 5000



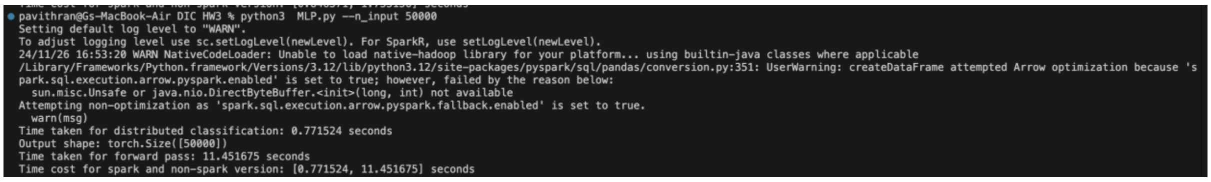
```
pavithran@Gs-MacBook-Air DIC HW3 % python3  MLP.py --n_input 5000
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/11/26 16:52:37 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/pyspark/sql/pandas/conversion.py:351: UserWarning: createDataFrame attempted Arrow optimization because 's
park.sql.execution.arrow.pyspark.enabled' is set to true; however, failed by the reason below:
  sun.misc.Unsafe or java.nio.DirectByteBuffer.<init>(long, int) not available
Attempting non-optimization as 'spark.sql.execution.arrow.pyspark.fallback.enabled' is set to true.
  warn(msg)
Time taken for distributed classification: 0.800821 seconds
Output shape: torch.Size([5000])
Time taken for forward pass: 0.887425 seconds
Time cost for spark and non-spark version: [0.800821, 0.887425] seconds
```

3) 10000



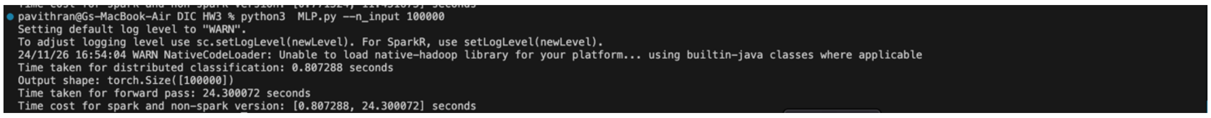
```
● pavithran@Gs-MacBook-Air DIC HW3 % python3  MLP.py --n_input 10000
  Setting default log level to "WARN".
  To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
  24/11/26 16:52:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
  /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/pyspark/sql/pandas/conversion.py:351: UserWarning: createDataFrame attempted Arrow optimization because 's
  park.sql.execution.arrow.pyspark.enabled' is set to true; however, failed by the reason below:
    sun.misc.Unsafe or java.nio.DirectByteBuffer.<init>(long, int) not available
  Attempting non-optimization as 'spark.sql.execution.arrow.pyspark.fallback.enabled' is set to true.
    warn(msg)
  Time taken for distributed classification: 0.846371 seconds
  Output shape: torch.Size([10000])
  Time taken for forward pass: 1.755156 seconds
  Time cost for spark and non-spark version: [0.846371, 1.755156] seconds
```

4) 50000



```
● pavithran@Gs-MacBook-Air DIC HW3 % python3  MLP.py --n_input 50000
  Setting default log level to "WARN".
  To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
  24/11/26 16:53:20 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
  /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/pyspark/sql/pandas/conversion.py:351: UserWarning: createDataFrame attempted Arrow optimization because 's
  park.sql.execution.arrow.pyspark.enabled' is set to true; however, failed by the reason below:
    sun.misc.Unsafe or java.nio.DirectByteBuffer.<init>(long, int) not available
  Attempting non-optimization as 'spark.sql.execution.arrow.pyspark.fallback.enabled' is set to true.
    warn(msg)
  Time taken for distributed classification: 0.771524 seconds
  Output shape: torch.Size([50000])
  Time taken for forward pass: 11.451675 seconds
  Time cost for spark and non-spark version: [0.771524, 11.451675] seconds
```

5) 100000



```
● pavithran@Gs-MacBook-Air DIC HW3 % python3  MLP.py --n_input 100000
  Setting default log level to "WARN".
  To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
  24/11/26 16:54:04 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
  Time taken for distributed classification: 0.807288 seconds
  Output shape: torch.Size([100000])
  Time taken for forward pass: 24.300072 seconds
  Time cost for spark and non-spark version: [0.807288, 24.300072] seconds
```

n_input	spark	Non spark
1000	0.883	0.189
5000	0.800	0.887
10000	0.846	1.755
50000	0.771	11.451
100000	0.8072	24.30

The key takeaway is how Spark handles larger workloads. For smaller inputs like 1,000, non-Spark has a slight edge in speed because the overhead of Spark's distributed processing isn't as significant. However, as the input size grows, non-Spark begins to struggle, taking much longer to process the data.

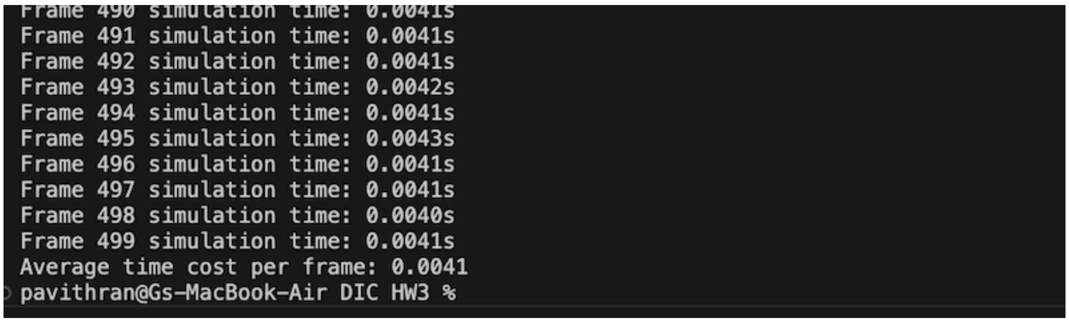
At an input size of 50,000, Spark demonstrates its efficiency by completing the task in just 0.771 seconds, while non-Spark takes 11.451 seconds. This massive difference highlights Spark's ability to efficiently distribute computations across resources, making it far more suited for handling large-scale datasets compared to non-Spark.

Task 1.3
you are required to run the simulation using both the Spark and non-Spark implementations with [200, 1,000, 5,000, 10,000] birds for 500

frames. Record the time cost per frame for both implementations and include these results in your report, along with a discussion of your observations.

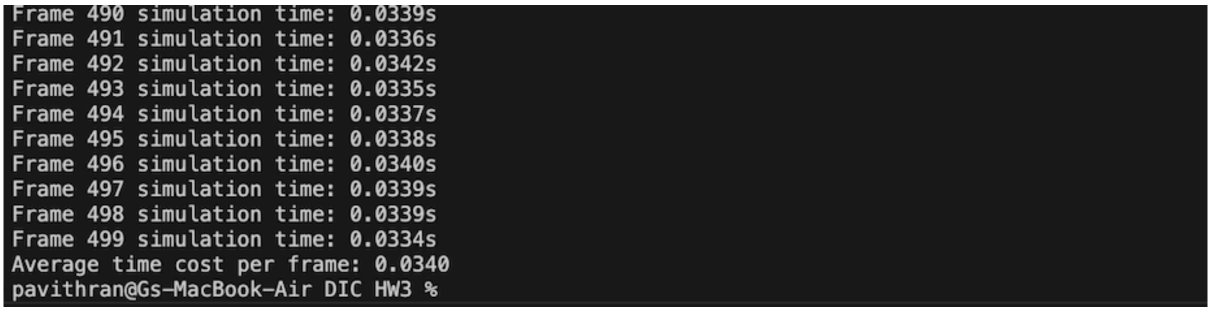
Spark Implementation

1) 200



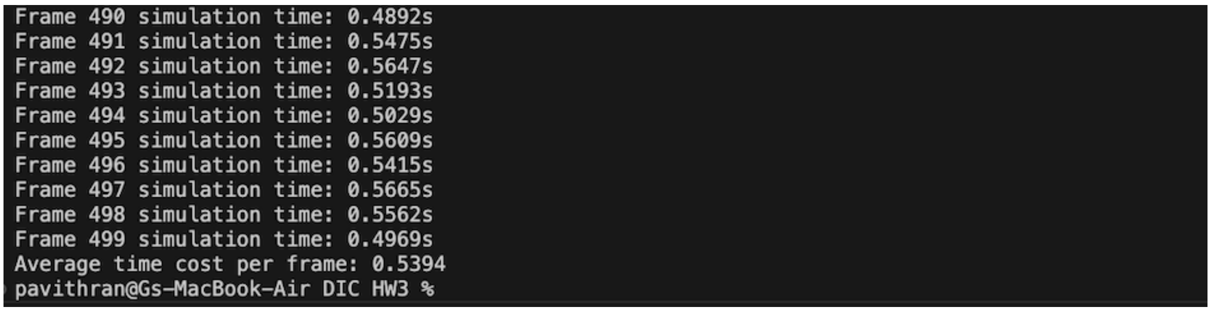
```
Frame 490 simulation time: 0.0041s
Frame 491 simulation time: 0.0041s
Frame 492 simulation time: 0.0041s
Frame 493 simulation time: 0.0042s
Frame 494 simulation time: 0.0041s
Frame 495 simulation time: 0.0043s
Frame 496 simulation time: 0.0041s
Frame 497 simulation time: 0.0041s
Frame 498 simulation time: 0.0040s
Frame 499 simulation time: 0.0041s
Average time cost per frame: 0.0041
pavithran@Gs-MacBook-Air DIC HW3 %
```

2) 1000



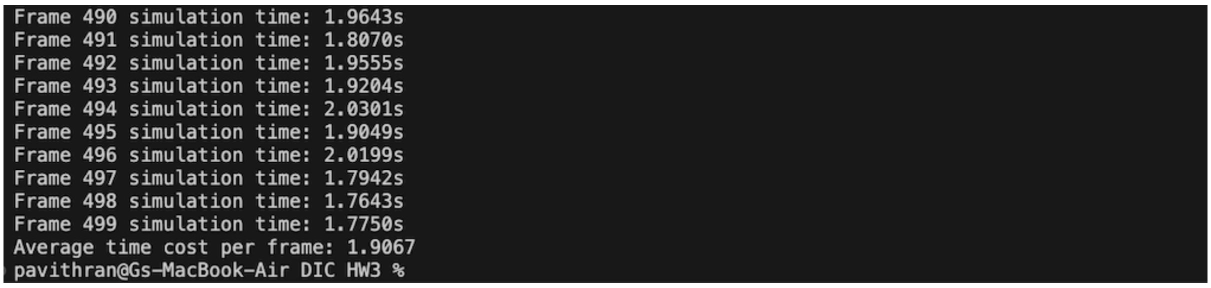
```
Frame 490 simulation time: 0.0339s
Frame 491 simulation time: 0.0336s
Frame 492 simulation time: 0.0342s
Frame 493 simulation time: 0.0335s
Frame 494 simulation time: 0.0337s
Frame 495 simulation time: 0.0338s
Frame 496 simulation time: 0.0340s
Frame 497 simulation time: 0.0339s
Frame 498 simulation time: 0.0339s
Frame 499 simulation time: 0.0334s
Average time cost per frame: 0.0340
pavithran@Gs-MacBook-Air DIC HW3 %
```

3) 5000



```
Frame 490 simulation time: 0.4892s
Frame 491 simulation time: 0.5475s
Frame 492 simulation time: 0.5647s
Frame 493 simulation time: 0.5193s
Frame 494 simulation time: 0.5029s
Frame 495 simulation time: 0.5609s
Frame 496 simulation time: 0.5415s
Frame 497 simulation time: 0.5665s
Frame 498 simulation time: 0.5562s
Frame 499 simulation time: 0.4969s
Average time cost per frame: 0.5394
pavithran@Gs-MacBook-Air DIC HW3 %
```

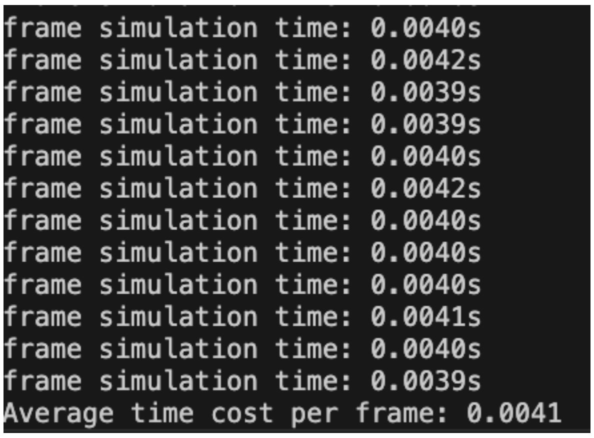
4) 10000



```
Frame 490 simulation time: 1.9643s
Frame 491 simulation time: 1.8070s
Frame 492 simulation time: 1.9555s
Frame 493 simulation time: 1.9204s
Frame 494 simulation time: 2.0301s
Frame 495 simulation time: 1.9049s
Frame 496 simulation time: 2.0199s
Frame 497 simulation time: 1.7942s
Frame 498 simulation time: 1.7643s
Frame 499 simulation time: 1.7750s
Average time cost per frame: 1.9067
pavithran@Gs-MacBook-Air DIC HW3 %
```

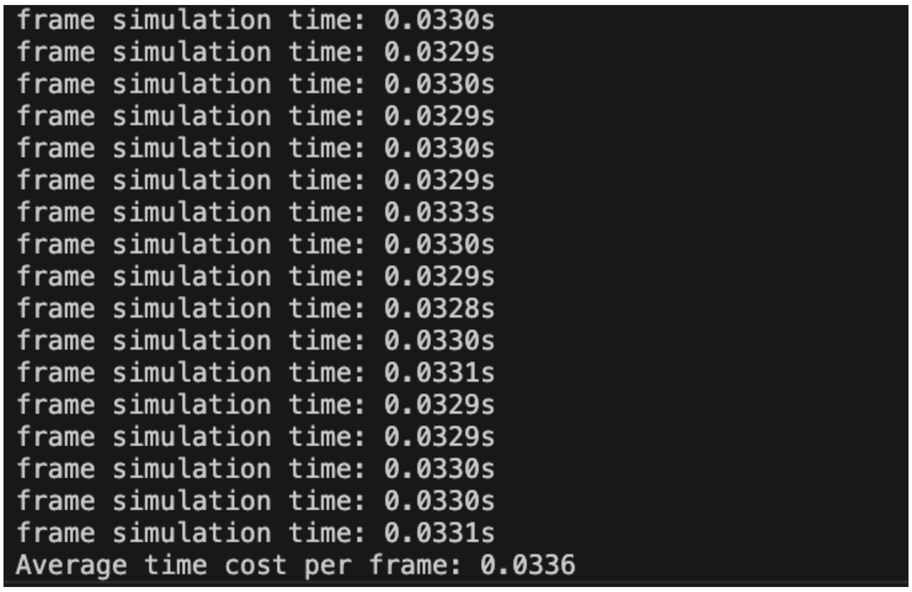
Non spark Implementation

1) 200



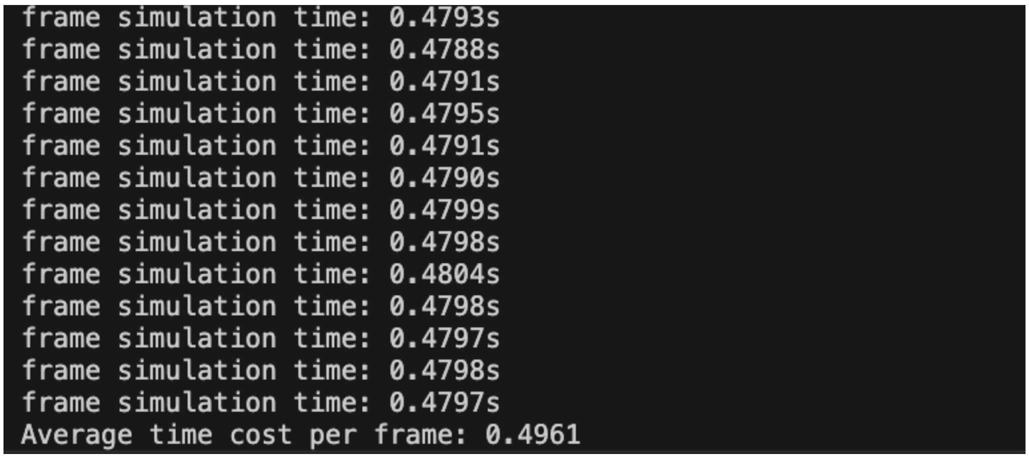
```
frame simulation time: 0.0040s
frame simulation time: 0.0042s
frame simulation time: 0.0039s
frame simulation time: 0.0039s
frame simulation time: 0.0040s
frame simulation time: 0.0042s
frame simulation time: 0.0040s
frame simulation time: 0.0040s
frame simulation time: 0.0040s
frame simulation time: 0.0041s
frame simulation time: 0.0040s
frame simulation time: 0.0039s
Average time cost per frame: 0.0041
```

2) 1000



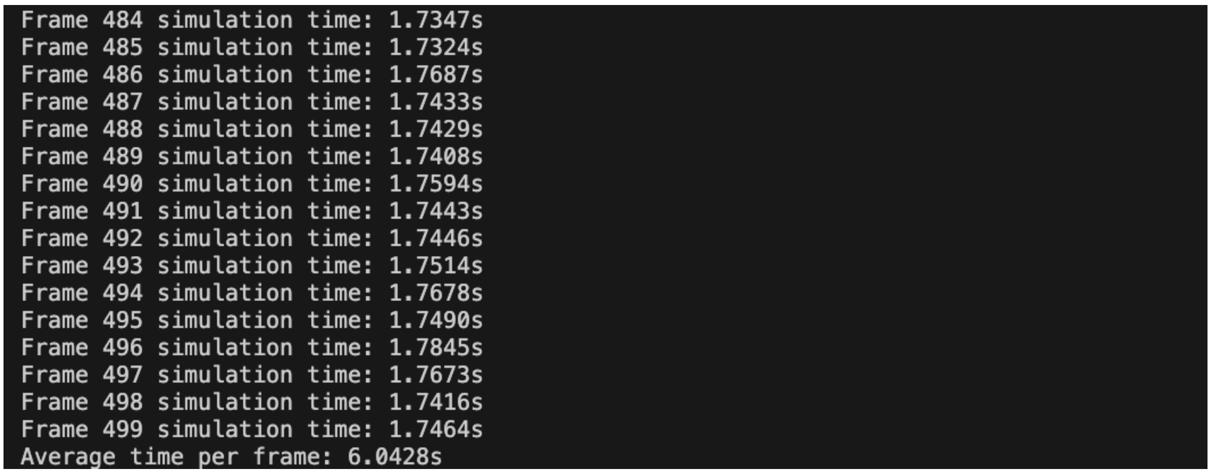
```
frame simulation time: 0.0330s
frame simulation time: 0.0329s
frame simulation time: 0.0330s
frame simulation time: 0.0329s
frame simulation time: 0.0330s
frame simulation time: 0.0329s
frame simulation time: 0.0333s
frame simulation time: 0.0330s
frame simulation time: 0.0329s
frame simulation time: 0.0328s
frame simulation time: 0.0330s
frame simulation time: 0.0331s
frame simulation time: 0.0329s
frame simulation time: 0.0329s
frame simulation time: 0.0330s
frame simulation time: 0.0330s
frame simulation time: 0.0331s
Average time cost per frame: 0.0336
```

3) 5000



```
frame simulation time: 0.4793s
frame simulation time: 0.4788s
frame simulation time: 0.4791s
frame simulation time: 0.4795s
frame simulation time: 0.4791s
frame simulation time: 0.4790s
frame simulation time: 0.4799s
frame simulation time: 0.4798s
frame simulation time: 0.4804s
frame simulation time: 0.4798s
frame simulation time: 0.4797s
frame simulation time: 0.4798s
frame simulation time: 0.4797s
Average time cost per frame: 0.4961
```

4) 10000



```
Frame 484 simulation time: 1.7347s
Frame 485 simulation time: 1.7324s
Frame 486 simulation time: 1.7687s
Frame 487 simulation time: 1.7433s
Frame 488 simulation time: 1.7429s
Frame 489 simulation time: 1.7408s
Frame 490 simulation time: 1.7594s
Frame 491 simulation time: 1.7443s
Frame 492 simulation time: 1.7446s
Frame 493 simulation time: 1.7514s
Frame 494 simulation time: 1.7678s
Frame 495 simulation time: 1.7490s
Frame 496 simulation time: 1.7845s
Frame 497 simulation time: 1.7673s
Frame 498 simulation time: 1.7416s
Frame 499 simulation time: 1.7464s
Average time per frame: 6.0428s
```

Number of birds	Spark (Average Time per Frame)	Non-Spark (Average Time per Frame)
200	0.0041	0.0041
1000	0.0340	0.0336
5000	0.5394	0.4961
10000	1.9067	6.0428

For small-scale simulations, such as with 200 or 1,000 birds, Spark and non-Spark implementations show nearly identical performance, averaging around 0.0041 and 0.034 seconds per frame, respectively. However, as the simulation scales up, differences start to appear. At 5,000 birds, Spark takes slightly longer (0.5394 seconds) compared to non-Spark (0.4961 seconds), but at 10,000 birds, Spark significantly outshines non-Spark, completing each frame in 1.9067 seconds versus the much slower 6.0428 seconds for non-Spark. This demonstrates that while non-Spark is efficient for smaller tasks, Spark's distributed architecture excels at handling larger datasets, offering substantial time savings at scale