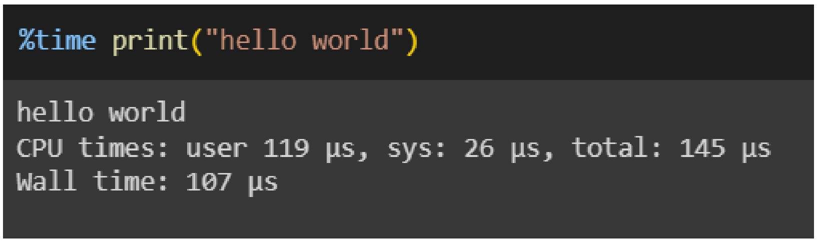
Intel optimizations benchmarking

Firltly I would like to explain how we performed the benchmarking , our technique and methods to achieve the similar benchmarking result

To analyze the execution of the cells in notebooks there are several options available but we choose the one mentioned in the inte oneapi samples, which is to use %time command with each code line or block we need to check time of ,for example



```
%time print("hello world")
```

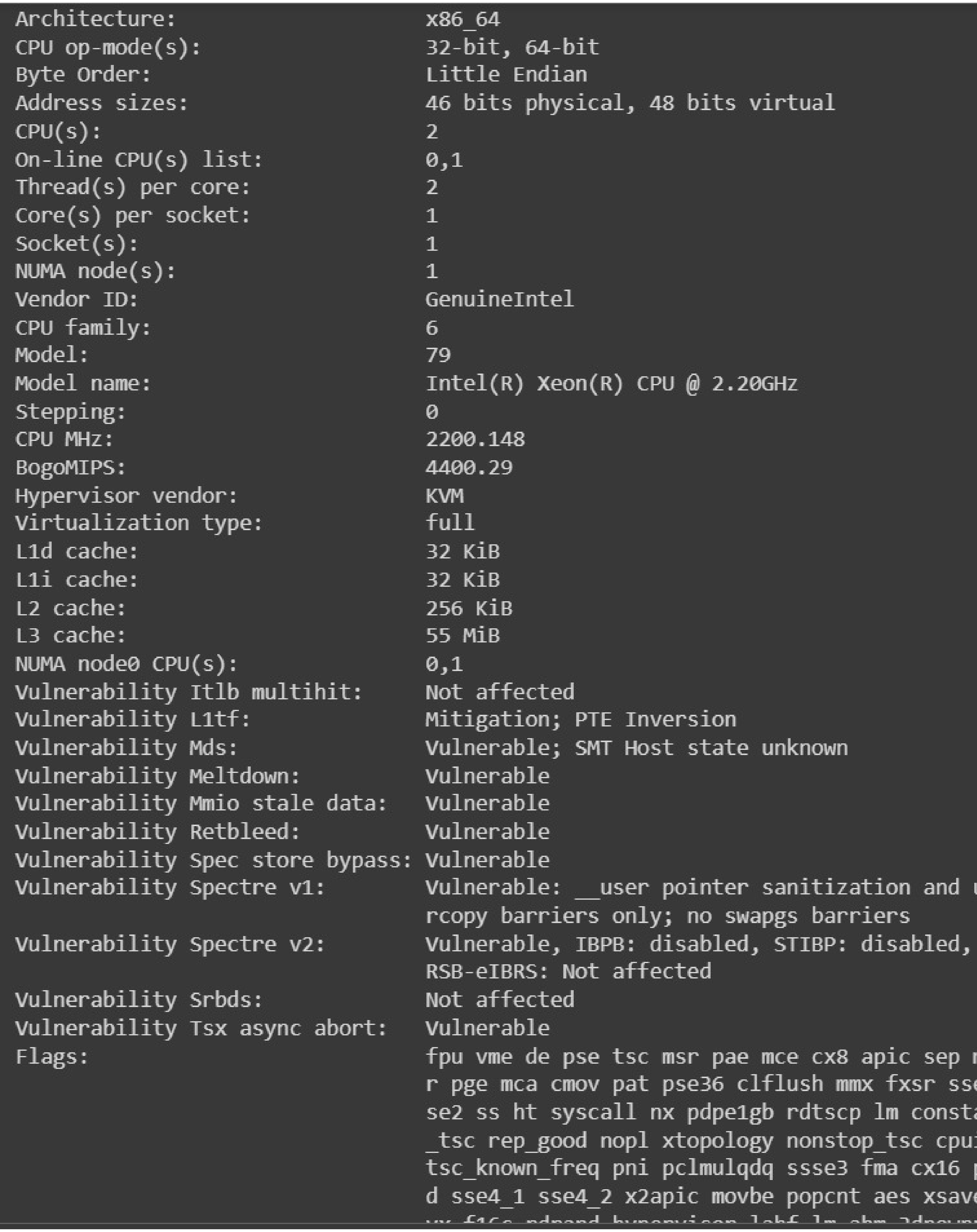
```
hello world
CPU times: user 119 µs, sys: 26 µs, total: 145 µs
Wall time: 107 µs
```

Architecture

Now I would like to explain about out running environment and architecture , we had several options like running the code offline ,using dev cloud or using collab , we choose the collab ,we know that running the code on devcloud is suggested but there we some know issues with it like runtime crash with modin, so we choose collab

Here is the Architecture details of cpu being used for benchmarking

To get these details you can run Lscpu command to get the detailed description of the cpu on which you are running code

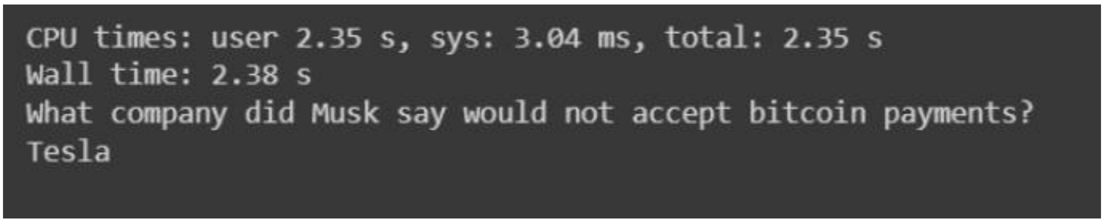


```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   46 bits physical, 48 bits virtual
CPU(s):                          2
On-line CPU(s) list:             0,1
Thread(s) per core:              2
Core(s) per socket:              1
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           79
Model name:                      Intel(R) Xeon(R) CPU @ 2.20GHz
Stepping:                        0
CPU MHz:                         2200.148
BogoMIPS:                        4400.29
Hypervisor vendor:               KVM
Virtualization type:             full
L1d cache:                       32 KiB
L1i cache:                       32 KiB
L2 cache:                        256 KiB
L3 cache:                        55 MiB
NUMA node0 CPU(s):               0,1
Vulnerability Itlb multihit:     Not affected
Vulnerability L1tf:              Mitigation; PTE Inversion
Vulnerability Mds:               Vulnerable; SMT Host state unknown
Vulnerability Meltdown:          Vulnerable
Vulnerability Mmio stale data:   Vulnerable
Vulnerability Retbleed:          Vulnerable
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1:        Vulnerable: __user pointer sanitization and u
                                 rcopy barriers only; no swapgs barriers
Vulnerability Spectre v2:        Vulnerable, IBPB: disabled, STIBP: disabled,
                                 RSB-eIBRS: Not affected
Vulnerability Srbds:             Not affected
Vulnerability Tsx async abort:   Vulnerable
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep m
                                 r pge mca cmov pat pse36 clflush mmx fxsr sse
                                 se2 ss ht syscall nx pdpe1gb rdtscp lm consta
                                 _tsc rep_good nopl xtopology nonstop_tsc cpui
                                 tsc_known_freq pni pclmulqdq ssse3 fma cx16 p
                                 d sse4_1 sse4_2 x2apic movbe popcnt aes xsave
                                 vx f16c rdrand hypervisor lahf_lm abm 3dnowpr
```

Benchmarking

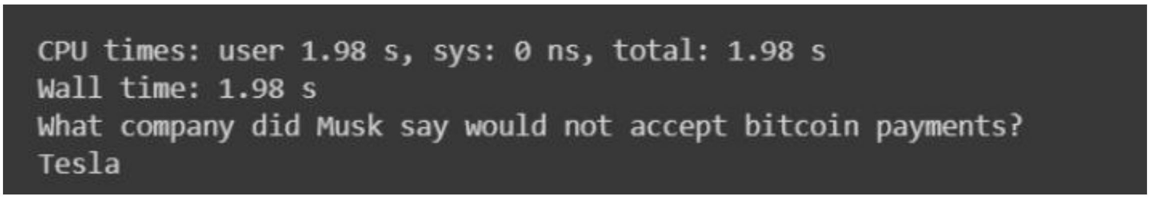
So here is the result we obtained by optimizing the pretrained models with ipex(intel optimization for pytorch)

Execution time without intel optimizations



```
CPU times: user 2.35 s, sys: 3.04 ms, total: 2.35 s
Wall time: 2.38 s
What company did Musk say would not accept bitcoin payments?
Tesla
```

Execution time with intel optimizations

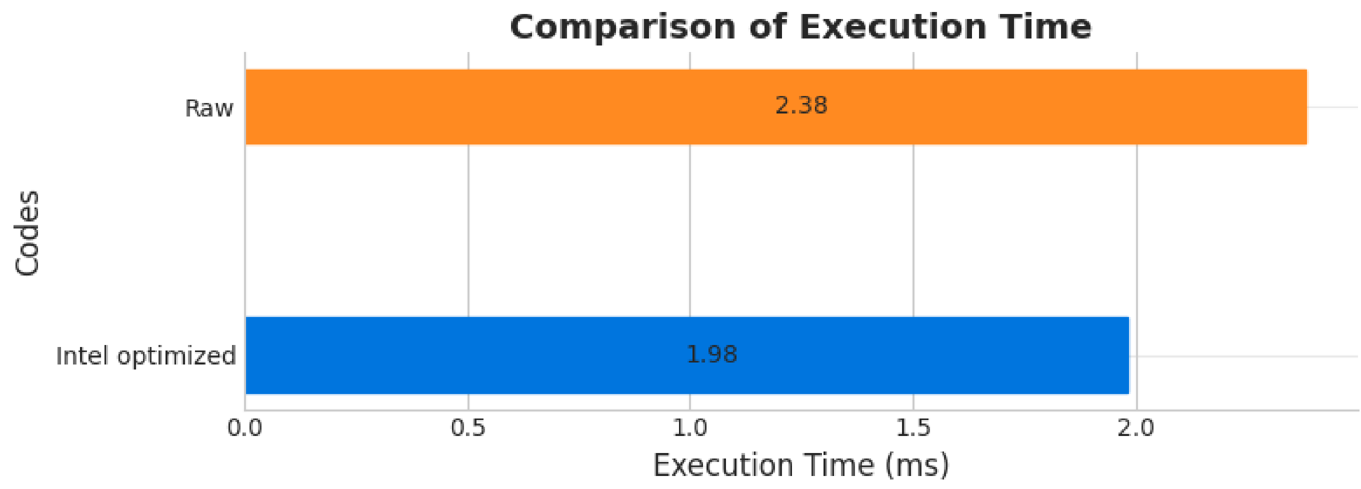


```
CPU times: user 1.98 s, sys: 0 ns, total: 1.98 s
Wall time: 1.98 s
What company did Musk say would not accept bitcoin payments?
Tesla
```

S = 2.38 ms / 1.98 ms ≈ 1.20

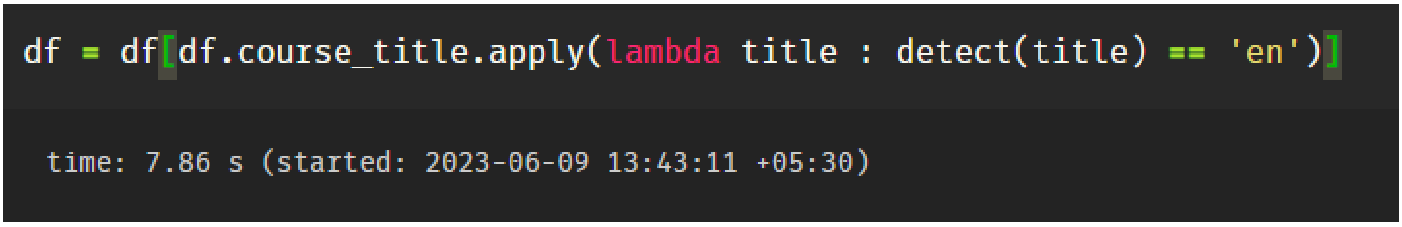
So, the second code run is approximately 1.20 times faster than the first code run.

In other words, the second code run is about 20% faster than the first code run.



here is the result we obtained by using modin instead of pandas

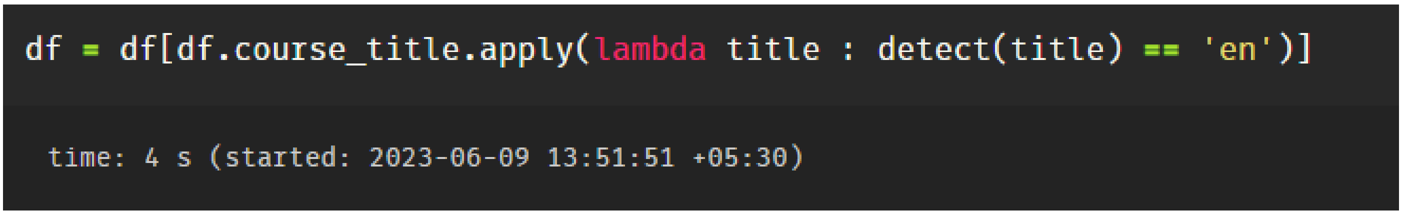
Execution time without intel optimizations



```
df = df[df.course_title.apply(lambda title : detect(title) == 'en')]

 time: 7.86 s (started: 2023-06-09 13:43:11 +05:30)
```

Execution time with intel optimizations



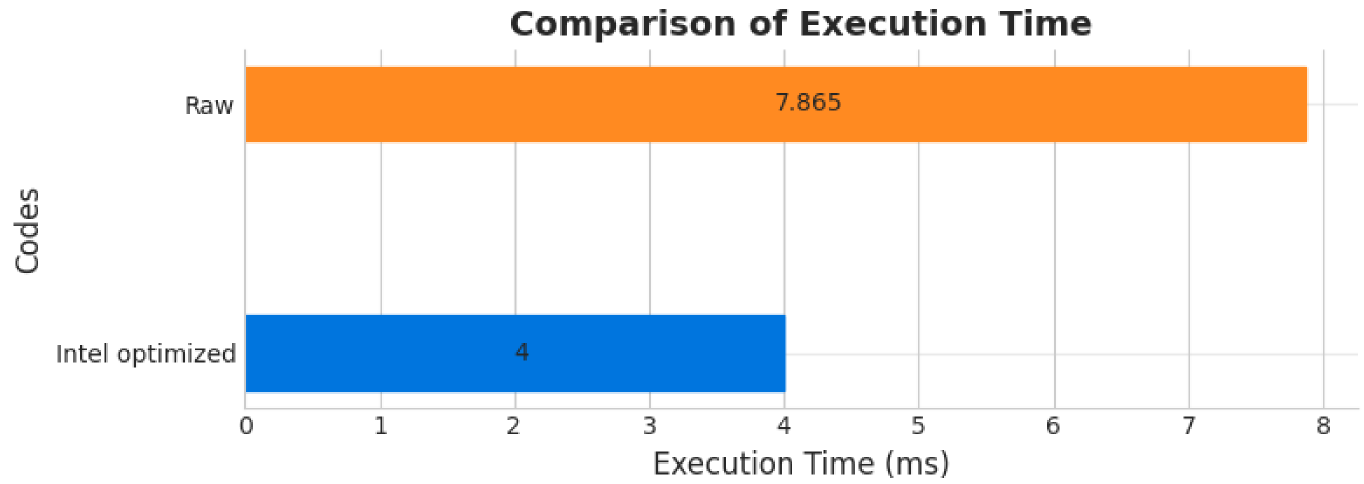
```
df = df[df.course_title.apply(lambda title : detect(title) == 'en')]

 time: 4 s (started: 2023-06-09 13:51:51 +05:30)
```

S = 7.865 ms / 4 ms ≈ 1.96625

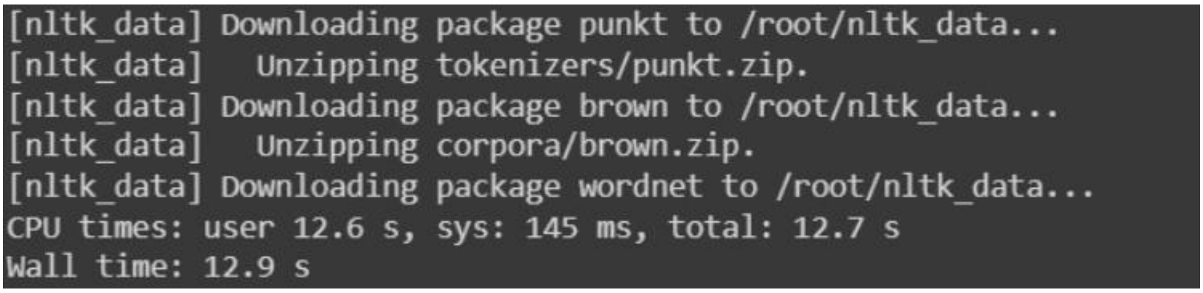
So, the second code run is approximately 1.96625 times faster than the first code run.

In other words, the second code run is about 96.625% faster than the first code run.



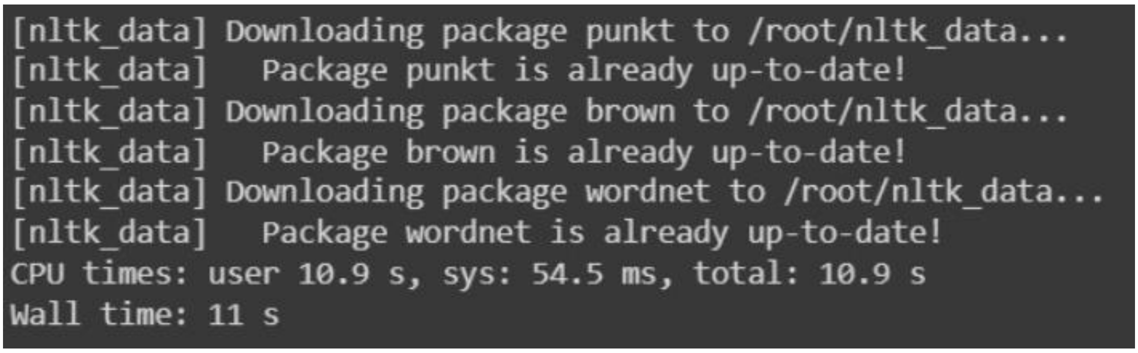
Summarization model

Code without intel optimizations



```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
CPU times: user 12.6 s, sys: 145 ms, total: 12.7 s
Wall time: 12.9 s
```

Code with intel optimizations

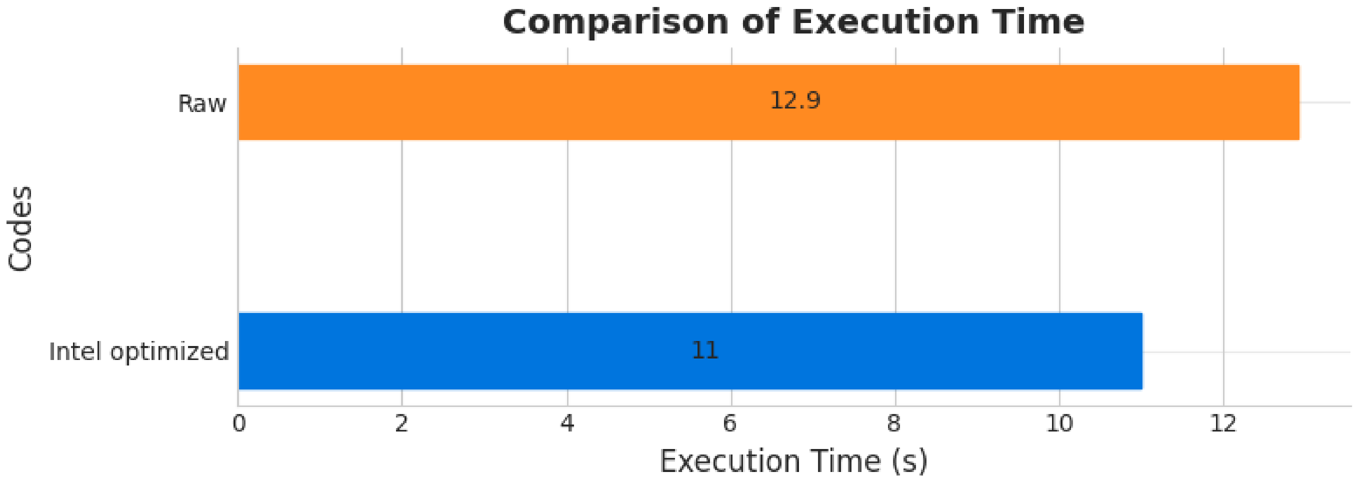


```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
CPU times: user 10.9 s, sys: 54.5 ms, total: 10.9 s
Wall time: 11 s
```

S = 12.9 ms / 11 ms ≈ 1.173

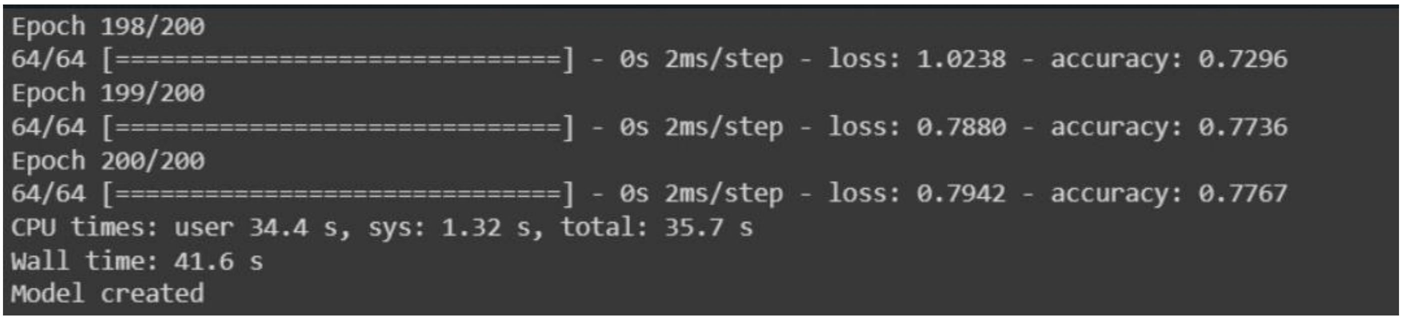
So, the second code run is approximately 1.173 times faster than the first code run.

In other words, the second code run is about 17.3% faster than the first code run



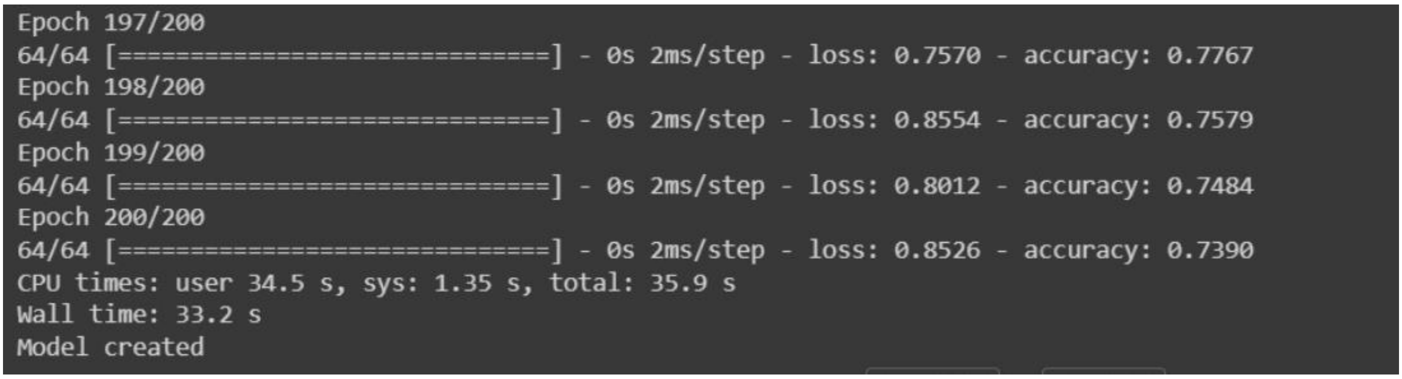
So here is the result we obtained by using intel optimisation for training the chat bot

Execution time before



```
Epoch 198/200
64/64 [==============================] - 0s 2ms/step - loss: 1.0238 - accuracy: 0.7296
Epoch 199/200
64/64 [==============================] - 0s 2ms/step - loss: 0.7880 - accuracy: 0.7736
Epoch 200/200
64/64 [==============================] - 0s 2ms/step - loss: 0.7942 - accuracy: 0.7767
CPU times: user 34.4 s, sys: 1.32 s, total: 35.7 s
Wall time: 41.6 s
Model created
```

Execution time after

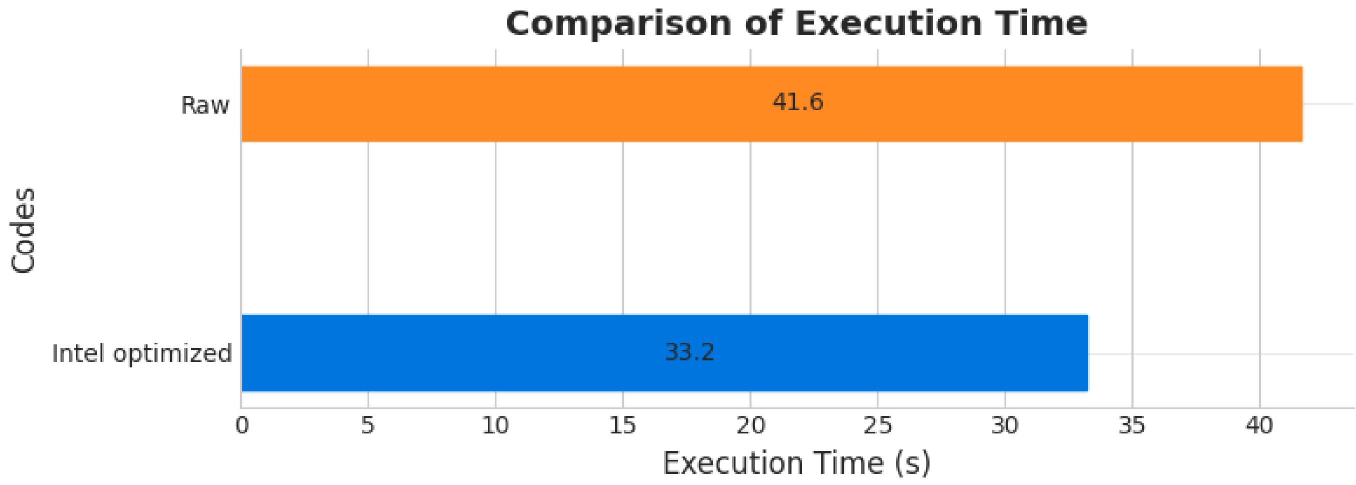


```
Epoch 197/200
64/64 [==============================] - 0s 2ms/step - loss: 0.7570 - accuracy: 0.7767
Epoch 198/200
64/64 [==============================] - 0s 2ms/step - loss: 0.8554 - accuracy: 0.7579
Epoch 199/200
64/64 [==============================] - 0s 2ms/step - loss: 0.8012 - accuracy: 0.7484
Epoch 200/200
64/64 [==============================] - 0s 2ms/step - loss: 0.8526 - accuracy: 0.7390
CPU times: user 34.5 s, sys: 1.35 s, total: 35.9 s
Wall time: 33.2 s
Model created
```

S = 41.6 s / 33.2 s ≈ 1.25

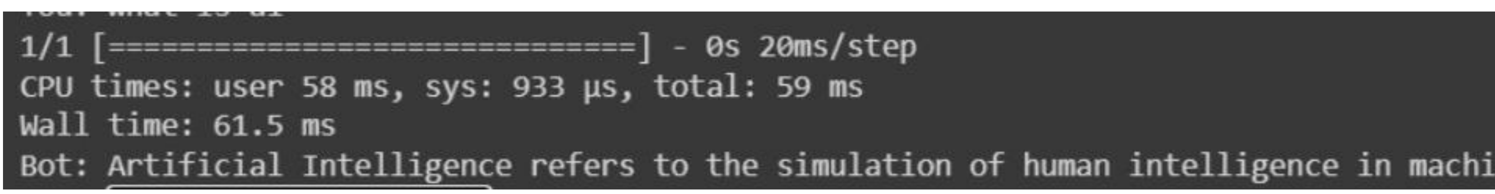
So, the second code run is approximately 1.25 times faster than the first code run.

In other words, the second code run is about 25% faster than the first code run.



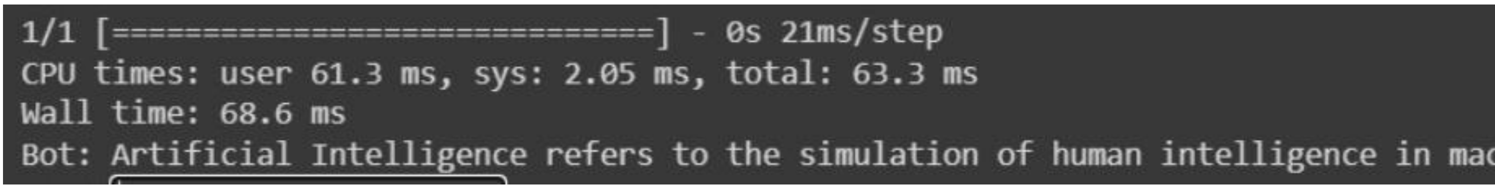
Chat bot Response time

Execution time before



```
1/1 [==============================] - 0s 20ms/step
CPU times: user 58 ms, sys: 933 µs, total: 59 ms
Wall time: 61.5 ms
Bot: Artificial Intelligence refers to the simulation of human intelligence in machi
```

Execution time after

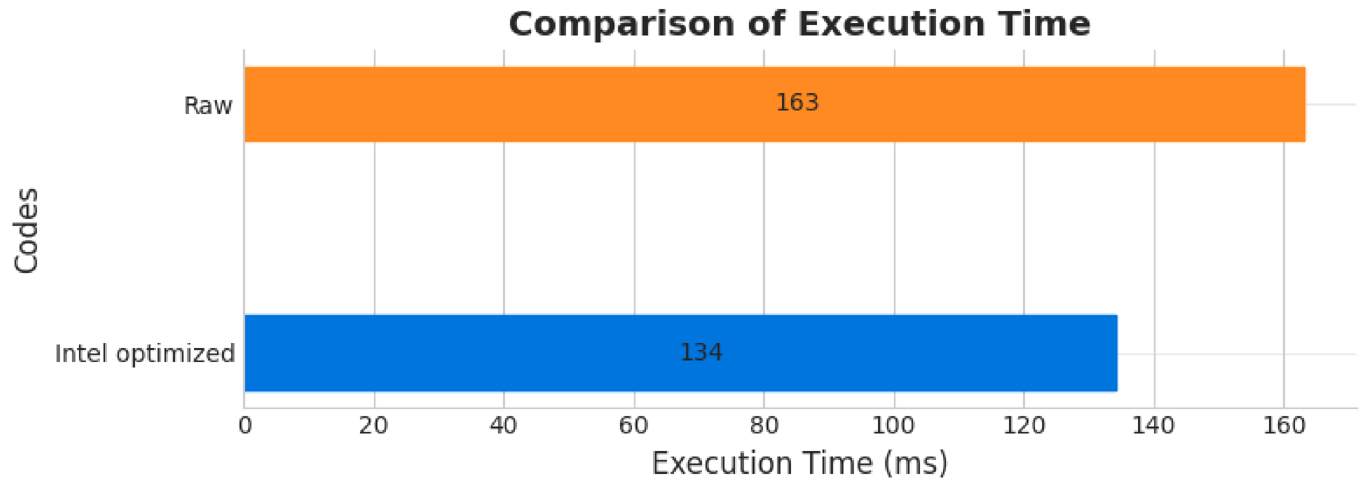


```
1/1 [==============================] - 0s 21ms/step
CPU times: user 61.3 ms, sys: 2.05 ms, total: 63.3 ms
Wall time: 68.6 ms
Bot: Artificial Intelligence refers to the simulation of human intelligence in mac
```

S = 68.6 ms / 61.5 ms ≈ 1.115

So, the second code run is approximately 1.115 times faster than the first code run.

In other words, the second code run is about 11.5% faster than the first code run.


Comparison of Execution Time
Raw
Intel optimized
Codes
Execution Time (ms)