

The background is a dark, teal-to-blue gradient. It features abstract, glowing circuitry patterns in white and light blue. Two prominent glowing nodes, resembling microchips or data centers, are positioned on the left and right sides, emitting light rays. The overall aesthetic is high-tech and digital.

powerpyro

Energy Consumption Monitoring
for Code Execution

Technical Documentation

API Reference

Módulo Monitor

Monitor

Class responsible for monitoring the energy consumption of selected hardware components (CPU, GPU, and memory) in a system.

Attributes:

Name	Type	Description
<code>__operating_system</code>	<code>OsType</code>	The current operating system.
<code>__components</code>	<code>Dict[str, HardwareComponent]</code>	Dictionary of initialized
<code>__stop_sign</code>	<code>bool</code>	Flag to stop the monitoring loop.
<code>__thread</code>	<code>Thread</code>	Thread in which monitoring occurs.
<code>__WATT_TO_KWH</code>	<code>float</code>	Constant to convert energy from

```
14 class Monitor():
15     """
16     Class responsible for monitoring the energy consumption of
17     selected hardware components
18     (CPU, GPU, and memory) in a system.
19
20     Attributes:
21         __operating_system (OsType): The current operating system.
22         __components (Dict[str, HardwareComponent]): Dictionary of
23 initialized
24         hardware components.
25         __stop_sign (bool): Flag to stop the monitoring loop.
26         __thread (Thread): Thread in which monitoring occurs.
27         __WATT_TO_KWH (float): Constant to convert energy from
28         watts to kilowatt-hours.
29     """
30     def __init__(self, required_components: Dict[str, bool]):
31         """
32         Initializes the Monitor class by setting up the operating system,
33         validating required components, creating component instances,
34         and preparing the monitoring thread.
35
36         Args:
37             required_components (Dict[str, bool]): Dictionary specifying
38 which
39             components ('cpu', 'gpu', 'memory') should be monitored.
40
41         Raises:
42             InvalidKeyError: If any invalid keys are found in
43             the provided dictionary.
44
45         Example:
46             Basic usage monitoring only CPU:
47
48             ```python
49             from power_pyro import Monitor
50
51             monitor = Monitor({'cpu': True, 'gpu': False, 'memory':
52 False})
53             ...
54
55             Monitoring CPU and GPU:
56
57             ```python
58             from power_pyro import Monitor
59
60             monitor = Monitor({'cpu': True, 'gpu': True, 'memory':
61 False})
62             ...
63
64             Monitoring all components:
65
66             ```python
67             from power_pyro import Monitor
68
69             monitor = Monitor({'cpu': True, 'gpu': True, 'memory': True})
70             ...
```

```

71         """
72         self.__operating_system: OsType = self.__get_operating_system()
73         self.__components: Dict[str, HardwareComponent] =
74     self.__create_components(required_components)
75         self.__stop_sign: bool = False
76         self.__thread: Thread = Thread(target=self.__monitor)
77         self.__WATT_TO_KWH: float = 3_600_000
78
79     def __get_operating_system(self) -> OsType:
80         """Determines the operating system type.
81
82         Returns:
83             The operating system type as OsType.
84
85         Raises:
86             OSError: If the OS cannot be identified.
87         """
88         if os.name == 'nt':
89             return OsType.WINDOWS
90         elif os.name == 'posix':
91             return OsType.LINUX
92         else:
93             raise OSError("Unable to identify operating system")
94
95     def get_monitored_components(self) -> Dict[str, Any]:
96         """Retrieves the components to be monitored.
97
98         Returns:
99             A dictionary with the monitoring status of the components
100             and the components themselves.
101
102         Example:
103             Example retrieving monitored components:
104
105             ```python
106             from power_pyro import Monitor
107
108             # Monitor only CPU and GPU
109             monitor = Monitor({'cpu': True, 'gpu': True, 'memory':
110 False})
111             components = monitor.get_monitored_components()
112
113             print(components)
114             # Output: {'cpu': True, 'gpu': True, 'memory': False}
115             ```
116         """
117         monitored_components: Dict[str, Any] = {'cpu': {'component': None,
118 'monitored': False},
119 'gpu': {'component': None,
120 'monitored': False},
121 'memory': {'component':
122 None, 'monitored': False}}
123
124         for key in self.__components.keys():
125             monitored_components[key]['component'] =
126 self.__components[key]
127             monitored_components[key]['monitored'] = True
128
129         return monitored_components
130
131     def __check_components(self, required_components: Dict[str, bool]) ->

```

```

132 bool:
133     """Validates the required components keys.
134
135     Args:
136         required_components: Dictionary of required components.
137
138     Returns:
139         bool:
140             - 'True' if all the dictionary keys are in the list,
141             - 'False' otherwise.
142     """
143     required_keys = ['cpu', 'gpu', 'memory']
144
145     return len(required_components.keys()) <= len(required_keys) and
146     all(key in required_keys for key in required_components)
147
148     def __create_components(self, required_components: Dict[str, bool]) -
149     > Dict[str, HardwareComponent]:
150         """Creates the required hardware components using the appropriate
151         factories.
152
153         Args:
154             required_components: Dictionary indicating which components
155             should be created.
156
157         Returns:
158             components: Dictionary containing the created hardware
159             components.
160
161         Raises:
162             InvalidKeysErrorException: If the required components contain
163             invalid keys.
164         """
165         if not self.__check_components(required_components):
166             raise InvalidKeysErrorException()
167
168         factories: Dict[str, HardwareComponentFactory] = {
169             'cpu': CpuComponentFactory(),
170             'gpu': GpuComponentFactory(),
171             'memory': MemoryComponentFactory()
172         }
173
174         components: Dict[str, HardwareComponent] = {}
175
176         for component in required_components:
177             components[component] =
178             factories[component].create_component(self.__operating_system)
179
180             if hasattr(components[component], 'open'):
181                 components[component].open()
182
183         return components
184
185     def __close_resources(self) -> None:
186         """Closes resources allocated by the components."""
187         for component in self.__components:
188
189             if hasattr(self.__components[component], 'close'):
190                 self.__components[component].close()
191
192     def get_energy_consumed_by_components(self) -> Dict[str, float]:

```

```

193         """Retrieves the total energy consumed by each hardware
194         component.
195
196         Returns:
197             energy_consumed_by_components: A dictionary where the keys
198             are component names ('cpu', 'gpu', 'memory') and the values are the
199             energy consumed by each component.
200
201         Example:
202             ```python
203             monitor = Monitor({'cpu': True, 'gpu': False})
204             result = monitor.get_energy_consumed_by_components()
205             print(result) # {'cpu': 2.5}
206             ```
207         """
208         energy_consumed_by_components: Dict[str, float] = {}
209
210         if 'cpu' in self.__components:
211             energy_consumed_by_components['cpu'] =
212             self.__components['cpu'].total_energy_consumed
213
214             if 'gpu' in self.__components:
215                 energy_consumed_by_components['gpu'] =
216                 self.__components['gpu'].total_energy_consumed
217
218                 if 'memory' in self.__components:
219                     energy_consumed_by_components['memory'] =
220                     self.__components['memory'].total_energy_consumed
221
222                     if 'memory' in self.__components:
223                         energy_consumed_by_components['memory'] =
224                         self.__components['memory'].total_energy_consumed
225
226                 return energy_consumed_by_components
227
228         def total_energy_consumed(self) -> float:
229             """Retrieves the total energy consumed by all components
230             monitored.
231
232             Example:
233                 Get total energy consumption after monitoring:
234
235                 ```python
236                 from power_pyro import Monitor
237
238                 monitor = Monitor({'cpu': True, 'gpu': True, 'memory':
239                 False})
240
241                 monitor.start()
242                 # ... perform operations ...
243                 monitor.end()
244
245                 total_energy = monitor.total_energy_consumed()
246                 print(f"Total energy consumed: {total_energy:.2f} Wh")
247                 ```
248             """
249
250             total_energy_consumed: float = 0.0
251
252             for component in self.__components:
253                 total_energy_consumed +=

```

```

254 self.__components[component].total_energy_consumed
255
256     return total_energy_consumed
257
258     def __monitor(self) -> None:
259         """Monitors energy consumption of components at regular
260 intervals."""
261
262         while not self.__stop_sign:
263             start = time.time()
264
265             time.sleep(10)
266
267             end = time.time()
268
269             period = end - start
270
271             if 'cpu' in self.__components:
272                 cpu = self.__components['cpu']
273                 cpu.update_energy_consumed((cpu.get_power() *
274 cpu.get_cpu_percent_for_process() * period)/self.__WATT_TO_KWH)
275
276             if 'gpu' in self.__components:
277                 gpu = self.__components['gpu']
278                 gpu.update_energy_consumed((gpu.get_power() *
279 period)/self.__WATT_TO_KWH)
280
281             if 'memory' in self.__components:
282                 memory = self.__components['memory']
283                 memory.update_energy_consumed((memory.get_power() *
284 period)/self.__WATT_TO_KWH)
285
286             if self.__operating_system == OsType.WINDOWS:
287                 self.__close_resources()
288
289     def start(self) -> None:
290         """
291         Starts the monitoring process in a separate thread.
292
293         Example:
294             Start the monitoring process:
295
296             ```python
297             from power_pyro import Monitor
298
299             monitor = Monitor({'cpu': True, 'gpu': False, 'memory':
300 True})
301             monitor.start()
302             ...
303         """
304         self.__thread.start()
305
306     def is_running(self) -> bool:
307         """
308         Checks if the monitoring process is currently running.
309
310         Returns:
311             bool: True if the monitoring thread is alive (running), False
312 otherwise.
313
314         Example:

```

```

Verify if the monitoring process is still running:

```python
from power_pyro import Monitor

monitor = Monitor({'cpu': True, 'gpu': False, 'memory':
True})

monitor.start()
print(monitor.is_running()) # True, since monitoring has
started

monitor.end()
print(monitor.is_running()) # False, since monitoring has
ended

...
"""
return self.__thread.is_alive()

def end(self) -> None:
 """
 Stops the monitoring process and waits for the monitoring thread
 to finish.

 Example:
 Stop the monitoring process:

    ```python
    from power_pyro import Monitor

    monitor = Monitor({'cpu': True, 'gpu': False, 'memory':
True})

    monitor.start()
    # ... perform operations to monitor ...
    monitor.end() # Stops monitoring and waits for thread to
finish

    ...
    """
    self.__stop_sign = True
    self.__thread.join()

```

`__check_components(required_components)`

Validates the required components keys.

Parameters:

Name	Type	Description	Default
<code>required_components</code>	<code>Dict[str, bool]</code>	Dictionary of required components.	<i>required</i>

Returns:

Name	Type	Description
<code>bool</code>	<code>bool</code>	<ul style="list-style-type: none"> • 'True' if all the dictionary keys are in the list, • 'False' otherwise.

” Source code in `power_pyro\monitor.py`

```

121 def __check_components(self, required_components: Dict[str, bool]) ->
122     bool:
123         """Validates the required components keys.
124
125         Args:
126             required_components: Dictionary of required components.
127
128         Returns:
129             bool:
130                 - 'True' if all the dictionary keys are in the list,
131                 - 'False' otherwise.
132         """
133         required_keys = ['cpu', 'gpu', 'memory']
134
135         return len(required_components.keys()) <= len(required_keys) and
136             all(key in required_keys for key in required_components)

```

`__close_resources()`

Closes resources allocated by the components.

” Source code in `power_pyro\monitor.py`

```

167 def __close_resources(self) -> None:
168     """Closes resources allocated by the components."""
169     for component in self.__components:
170
171         if hasattr(self.__components[component], 'close'):
172             self.__components[component].close()

```

`__create_components(required_components)`

Creates the required hardware components using the appropriate factories.

Parameters:

Name	Type	Description	Default
<code>required_components</code>	<code>Dict[str, bool]</code>	Dictionary indicating which components should be created.	<i>required</i>

Returns:

Name	Type	Description
<code>components</code>	<code>Dict[str, HardwareComponent]</code>	Dictionary containing the created hardware components.

Raises:

Type	Description
<code>InvalidKeysErrorException</code>	If the required components contain invalid keys.

Source code in power_pyro\monitor.py

```
136 def __create_components(self, required_components: Dict[str, bool]) ->
137 Dict[str, HardwareComponent]:
138     """Creates the required hardware components using the appropriate
139     factories.
140
141     Args:
142         required_components: Dictionary indicating which components
143         should be created.
144
145     Returns:
146         components: Dictionary containing the created hardware
147         components.
148
149     Raises:
150         InvalidKeyError: If the required components contain
151         invalid keys.
152     """
153     if not self.__check_components(required_components):
154         raise InvalidKeyError()
155
156     factories: Dict[str, HardwareComponentFactory] = {
157         'cpu': CpuComponentFactory(),
158         'gpu': GpuComponentFactory(),
159         'memory': MemoryComponentFactory()
160     }
161
162     components: Dict[str, HardwareComponent] = {}
163
164     for component in required_components:
165         components[component] =
            factories[component].create_component(self.__operating_system)
            if hasattr(components[component], 'open'):
                components[component].open()
166
167     return components
```

__get_operating_system()

Determines the operating system type.

Returns:

Type	Description
OsType	The operating system type as OsType.

Raises:

Type	Description
<code>OSError</code>	If the OS cannot be identified.

Source code in `power_pyro\monitor.py`

```
74 def __get_operating_system(self) -> OsType:
75     """Determines the operating system type.
76
77     Returns:
78         The operating system type as OsType.
79
80     Raises:
81         OSErrors: If the OS cannot be identified.
82     """
83     if os.name == 'nt':
84         return OsType.WINDOWS
85     elif os.name == 'posix':
86         return OsType.LINUX
87     else:
88         raise OSErrors("Unable to identify operating system")
```

`__init__(required_components)`

Initializes the Monitor class by setting up the operating system, validating required components, creating component instances, and preparing the monitoring thread.

Parameters:

Name	Type	Description	Default
<code>required_components</code>	<code>Dict[str, bool]</code>	Dictionary specifying which	<i>required</i>

Raises:

Type	Description
<code>InvalidKeysErrorException</code>	If any invalid keys are found in

Example



Basic usage monitoring only CPU:

```
from power_pyro import Monitor  
  
monitor = Monitor({'cpu': True, 'gpu': False, 'memory': False})
```

Monitoring CPU and GPU:

```
from power_pyro import Monitor  
  
monitor = Monitor({'cpu': True, 'gpu': True, 'memory': False})
```

Monitoring all components:

```
from power_pyro import Monitor  
  
monitor = Monitor({'cpu': True, 'gpu': True, 'memory': True})
```

Source code in power_pyro\monitor.py

```
29 def __init__(self, required_components: Dict[str, bool]):
30     """
31     Initializes the Monitor class by setting up the operating system,
32     validating required components, creating component instances,
33     and preparing the monitoring thread.
34
35     Args:
36         required_components (Dict[str, bool]): Dictionary specifying which
37         components ('cpu', 'gpu', 'memory') should be monitored.
38
39     Raises:
40         InvalidKeyError: If any invalid keys are found in
41         the provided dictionary.
42
43     Example:
44         Basic usage monitoring only CPU:
45
46         ```python
47         from power_pyro import Monitor
48
49         monitor = Monitor({'cpu': True, 'gpu': False, 'memory': False})
50         ```
51
52         Monitoring CPU and GPU:
53
54         ```python
55         from power_pyro import Monitor
56
57         monitor = Monitor({'cpu': True, 'gpu': True, 'memory': False})
58         ```
59
60         Monitoring all components:
61
62         ```python
63         from power_pyro import Monitor
64
65         monitor = Monitor({'cpu': True, 'gpu': True, 'memory': True})
66         ```
67     """
68     self.__operating_system: OsType = self.__get_operating_system()
69     self.__components: Dict[str, HardwareComponent] =
70 self.__create_components(required_components)
71     self.__stop_sign: bool = False
72     self.__thread: Thread = Thread(target=self.__monitor)
73     self.__WATT_TO_KWH: float = 3_600_000
```

__monitor()

Monitors energy consumption of components at regular intervals.

Source code in `power_pyro\monitor.py`

```
230 def __monitor(self) -> None:
231     """Monitors energy consumption of components at regular intervals."""
232
233     while not self.__stop_sign:
234         start = time.time()
235
236         time.sleep(10)
237
238         end = time.time()
239
240         period = end - start
241
242         if 'cpu' in self.__components:
243             cpu = self.__components['cpu']
244             cpu.update_energy_consumed((cpu.get_power() *
245 cpu.get_cpu_percent_for_process() * period)/self.__WATT_TO_KWH)
246
247         if 'gpu' in self.__components:
248             gpu = self.__components['gpu']
249             gpu.update_energy_consumed((gpu.get_power() *
250 period)/self.__WATT_TO_KWH)
251
252         if 'memory' in self.__components:
253             memory = self.__components['memory']
254             memory.update_energy_consumed((memory.get_power() *
255 period)/self.__WATT_TO_KWH)
256
257         if self.__operating_system == OsType.WINDOWS:
258             self.__close_resources()
```

`end()`

Stops the monitoring process and waits for the monitoring thread to finish.

Example

Stop the monitoring process:

```
from power_pyro import Monitor

monitor = Monitor({'cpu': True, 'gpu': False, 'memory': True})
monitor.start()
# ... perform operations to monitor ...
monitor.end() # Stops monitoring and waits for thread to finish
```

Source code in `power_pyro\monitor.py`

```
295 def end(self) -> None:
296     """
297     Stops the monitoring process and waits for the monitoring thread to
298     finish.
299
300     Example:
301         Stop the monitoring process:
302
303         ```python
304         from power_pyro import Monitor
305
306         monitor = Monitor({'cpu': True, 'gpu': False, 'memory': True})
307         monitor.start()
308         # ... perform operations to monitor ...
309         monitor.end() # Stops monitoring and waits for thread to finish
310         ```
311     """
312     self.__stop_sign = True
    self.__thread.join()
```

`get_energy_consumed_by_components()`

Retrieves the total energy consumed by each hardware component.

Returns:

Name	Type	Description
<code>energy_consumed_by_components</code>	<code>Dict[str, float]</code>	A dictionary where the keys are component names ('cpu', 'gpu', 'memory') and the values are the energy consumed by each component.

Example

```
monitor = Monitor({'cpu': True, 'gpu': False})
result = monitor.get_energy_consumed_by_components()
print(result) # {'cpu': 2.5}
```


Source code in `power_pyro\monitor.py`

```
174 def get_energy_consumed_by_components(self) -> Dict[str, float]:
175     """Retrieves the total energy consumed by each hardware component.
176
177     Returns:
178         energy_consumed_by_components: A dictionary where the keys are
179         component names ('cpu', 'gpu', 'memory') and the values are the energy
180         consumed by each component.
181
182     Example:
183         ```python
184         monitor = Monitor({'cpu': True, 'gpu': False})
185         result = monitor.get_energy_consumed_by_components()
186         print(result) # {'cpu': 2.5}
187         ```
188     """
189     energy_consumed_by_components: Dict[str, float] = {}
190
191     if 'cpu' in self.__components:
192         energy_consumed_by_components['cpu'] =
193         self.__components['cpu'].total_energy_consumed
194
195     if 'gpu' in self.__components:
196         energy_consumed_by_components['gpu'] =
197         self.__components['gpu'].total_energy_consumed
198
199     if 'memory' in self.__components:
200         energy_consumed_by_components['memory'] =
201         self.__components['memory'].total_energy_consumed
202
203     if 'memory' in self.__components:
204         energy_consumed_by_components['memory'] =
205         self.__components['memory'].total_energy_consumed
206
207     return energy_consumed_by_components
```

`get_monitored_components()`

Retrieves the components to be monitored.

Returns:

Type	Description
<code>Dict[str, Any]</code>	A dictionary with the monitoring status of the components
<code>Dict[str, Any]</code>	and the components themselves.

Example

Example retrieving monitored components:

```
from power_pyro import Monitor

# Monitor only CPU and GPU
monitor = Monitor({'cpu': True, 'gpu': True, 'memory': False})
components = monitor.get_monitored_components()

print(components)
# Output: {'cpu': True, 'gpu': True, 'memory': False}
```

Source code in `power_pyro\monitor.py`

```
90 def get_monitored_components(self) -> Dict[str, Any]:
91     """Retrieves the components to be monitored.
92
93     Returns:
94         A dictionary with the monitoring status of the components
95         and the components themselves.
96
97     Example:
98         Example retrieving monitored components:
99
100         ```python
101         from power_pyro import Monitor
102
103         # Monitor only CPU and GPU
104         monitor = Monitor({'cpu': True, 'gpu': True, 'memory': False})
105         components = monitor.get_monitored_components()
106
107         print(components)
108         # Output: {'cpu': True, 'gpu': True, 'memory': False}
109         ```
110     """
111     monitored_components: Dict[str, Any] = {'cpu': {'component': None,
112 'monitored': False},
113                                             'gpu': {'component': None,
114 'monitored': False},
115                                             'memory': {'component': None,
116 'monitored': False}}
117
118     for key in self.__components.keys():
119         monitored_components[key]['component'] = self.__components[key]
120         monitored_components[key]['monitored'] = True
121
122     return monitored_components
```

`is_running()`

Checks if the monitoring process is currently running.

Returns:

Name	Type	Description
<code>bool</code>	<code>bool</code>	True if the monitoring thread is alive (running), False otherwise.



Example



Verify if the monitoring process is still running:

```
from power_pyro import Monitor

monitor = Monitor({'cpu': True, 'gpu': False, 'memory': True})
monitor.start()
print(monitor.is_running()) # True, since monitoring has started
monitor.end()
print(monitor.is_running()) # False, since monitoring has ended
```



Source code in `power_pyro\monitor.py`



```
273 def is_running(self) -> bool:
274     """
275     Checks if the monitoring process is currently running.
276
277     Returns:
278         bool: True if the monitoring thread is alive (running), False
279         otherwise.
280
281     Example:
282         Verify if the monitoring process is still running:
283
284         ```python
285         from power_pyro import Monitor
286
287         monitor = Monitor({'cpu': True, 'gpu': False, 'memory': True})
288         monitor.start()
289         print(monitor.is_running()) # True, since monitoring has started
290         monitor.end()
291         print(monitor.is_running()) # False, since monitoring has ended
292         ```
293     """
    return self.__thread.is_alive()
```

`start()`

Starts the monitoring process in a separate thread.



Example



Start the monitoring process:

```
from power_pyro import Monitor

monitor = Monitor({'cpu': True, 'gpu': False, 'memory': True})
monitor.start()
```



Source code in `power_pyro\monitor.py`



```
257 def start(self) -> None:
258     """
259     Starts the monitoring process in a separate thread.
260
261     Example:
262     Start the monitoring process:
263
264     ```python
265     from power_pyro import Monitor
266
267     monitor = Monitor({'cpu': True, 'gpu': False, 'memory': True})
268     monitor.start()
269     ```
270     """
271     self.__thread.start()
```

`total_energy_consumed()`

Retrieves the total energy consumed by all components monitored.



Example



Get total energy consumption after monitoring:

```
from power_pyro import Monitor

monitor = Monitor({'cpu': True, 'gpu': True, 'memory': False})
monitor.start()
# ... perform operations ...
monitor.end()

total_energy = monitor.total_energy_consumed()
print(f"Total energy consumed: {total_energy:.2f} Wh")
```

Source code in power_pyro\monitor.py

```
203 def total_energy_consumed(self) -> float:
204     """Retrieves the total energy consumed by all components monitored.
205
206     Example:
207         Get total energy consumption after monitoring:
208
209         ```python
210         from power_pyro import Monitor
211
212         monitor = Monitor({'cpu': True, 'gpu': True, 'memory': False})
213         monitor.start()
214         # ... perform operations ...
215         monitor.end()
216
217         total_energy = monitor.total_energy_consumed()
218         print(f"Total energy consumed: {total_energy:.2f} Wh")
219         ```
220
221     """
222
223     total_energy_consumed: float = 0.0
224
225     for component in self.__components:
226         total_energy_consumed +=
227     self.__components[component].total_energy_consumed
228
229     return total_energy_consumed
```

Módulo CPU

Cpu

Bases: ProcessingUnit

Represents a Central Processing Unit (CPU) responsible for accessing and retrieving power consumption values from hardware sensors.

Attributes:

Name	Type	Description
__manufacturer	CpuType	CPU type.

```

26 class Cpu(ProcessingUnit):
27     """Represents a Central Processing Unit (CPU) responsible
28         for accessing and retrieving power consumption values
29         from hardware sensors.
30
31     Attributes:
32         __manufacturer (CpuType): CPU type.
33     """
34     def __init__(self, operating_system: OsType):
35         super().__init__(operating_system)
36         self.__manufacturer: CpuType
37
38         if operating_system == OsType.WINDOWS:
39             self.computer.IsCpuEnabled = True
40
41         self._update_manufacture()
42         self._update_hardware_name()
43
44     @property
45     def get_manufacturer(self) -> CpuType:
46         """ The hardware manufacturer.
47
48     Returns:
49         CpuType: A type of CPU.
50
51     Example:
52         ```python
53
54         from monitor import Monitor
55
56         monitor = Monitor({'cpu': True})
57
58         components = monitor.get_monitored_components()
59         print(components['cpu']['component'].get_manufacturer)
60 #CpuType.INTEL
61
62         ...
63         """
64         return self.__manufacturer
65
66     def _update_manufacture(self) -> None:
67         if self.operating_system == OsType.WINDOWS:
68             self._update_manufacture_windows()
69
70         elif self.operating_system == OsType.LINUX:
71             self._update_manufacture_linux()
72         else:
73             raise OSError("Unable to identify operating system")
74
75     def _update_manufacture_windows(self) -> None:
76         """Update hardware manufacturer when running on Windows OS.
77
78     Raises:
79         IdentifyHardwareManufacturerException: If the hardware
80     manufacturer
81         cannot be identified.
82         """

```

```

83
84     try:
85         wmi_session = wmi.WMI()
86
87         manufacturer = wmi_session.Win32_Processor()[0].Manufacturer
88
89         if manufacturer == 'GenuineIntel':
90             self.__manufacturer = CpuType.INTEL
91         elif manufacturer == 'AuthenticAMD':
92             self.__manufacturer = CpuType.AMD
93         else:
94             raise IdentifyHardwareManufacturerException(HT.CPU)
95     except (ModuleNotFoundError, wmi.x_wmi, IndexError,
96 AttributeError):
97         raise IdentifyHardwareManufacturerException(HT.CPU)
98
99     def __update_manufacture_linux(self) -> None:
100         """Update hardware manufacturer when running on Linux OS.
101
102         Raises:
103             IdentifyHardwareManufacturerException: If the hardware
104 manufacturer
105             cannot be identified.
106         """
107
108         try:
109             info = cpuinfo.get_cpu_info()
110
111             manufacturer = info['vendor_id_raw']
112
113             if manufacturer == 'GenuineIntel':
114                 self.__manufacturer = CpuType.INTEL
115             elif manufacturer == 'AuthenticAMD':
116                 self.__manufacturer = CpuType.AMD
117             else:
118                 raise IdentifyHardwareManufacturerException(HT.CPU)
119         except (ModuleNotFoundError, KeyError):
120             raise IdentifyHardwareManufacturerException(HT.CPU)
121
122     def _update_hardware_name(self) -> None:
123         if self.operating_system == OsType.WINDOWS:
124             self.__update_hardware_name_windows()
125         elif self.operating_system == OsType.LINUX:
126             self.__update_hardware_name_linux()
127         else:
128             raise OSError("Unable to identify operating system")
129
130     def __update_hardware_name_windows(self) -> None:
131         """ Set the CPU name.
132
133         Raises:
134             HardwareNameIdentifyException: Unable to identify CPU name in
135 Windows.
136         """
137
138         try:
139             wmi_session = wmi.WMI()
140
141             self.set_name = wmi_session.Win32_Processor()[0].Name
142         except (ModuleNotFoundError, wmi.x_wmi, IndexError,
143 AttributeError):

```

```

144         raise HardwareNameIdentifyException(HT.CPU)
145
146     def __update_hardware_name_linux(self) -> None:
147         """ Set the CPU name.
148
149         Raises:
150             HardwareNameIdentifyException: Unable to identify CPU name in
151 Linux.
152         """
153
154         try:
155             self.set_name = cpuinfo.get_cpu_info()['brand_raw']
156         except (ModuleNotFoundError, KeyError):
157             raise HardwareNameIdentifyException(HT.CPU)
158
159     def get_power(self) -> float:
160         if self.operating_system == OsType.WINDOWS:
161             return self.__get_power_on_windows()
162
163         else:
164             return self.__get_power_on_linux()
165
166     def __get_power_on_linux(self) -> float:
167         """ Returns the value of the CPU power in W in Linux.
168
169         Returns:
170             float: CPU power.
171         """
172
173         try:
174             command = ["sudo", "perf", "stat", "-e", "power/energy-pkg/",
175 "sleep", "0.1"]
176             power = subprocess.run(command, capture_output=True,
177 text=True)
178
179             power = power.stderr.split(" ")
180             power = [string for string in power if string.strip()]
181
182             for index, string in enumerate(power):
183                 if string.find('\n\n') != -1:
184                     power = power[index + 1]
185                     power = power.replace(",", ".")
186                     break
187
188             power = float(power)/0.1
189         except (PermissionError, subprocess.SubprocessError,
190 AttributeError, IndexError, ValueError) as e:
191             print('Error getting power from CPU: ', str(e))
192
193         return power
194
195     def __get_power_on_windows(self) -> float:
196         """ Returns the value of the CPU power in W in Windows.
197
198         Returns:
199             float: CPU power.
200         """
201
202         try:
203             cpu = next((hardware for hardware in self.computer.Hardware
204 if hardware.HardwareType == HardwareType.Cpu), None)
205             cpu.Update()

```



```

205         time.sleep(0.1)
206
207         power = next((sensor for sensor in cpu.Sensors if
208 sensor.SensorType == SensorType.Power and (sensor.Name == "CPU Package"
209 or sensor.Name == "Package")))
210         power = power.Value
211         except AttributeError as e:
212             print('Error getting power from CPU: ', str(e))
213
214         return power
215
216     def get_cpu_percent_for_process(self) -> float:
217         """ Returns the percentage value of the monitored process on the
218 CPU.
219
220         Returns:
221             float: CPU percent.
222
223         Example:
224             ```python
225
226             from monitor import Monitor
227
228             monitor = Monitor({'cpu': True})
229             components = monitor.get_monitored_components()
230             print(components['cpu'])
231 ['component'].get_cpu_percent_for_process()) # 0.37
232             ```
233         """
234         script_pid = os.getpid()
235         sum_all = 0
236         cpu_percent = 0
237         for process in psutil.process_iter():
238             try:
239                 with process.oneshot():
240                     process_pid = process.pid
241                     process_cpu_percent = process.cpu_percent()
242                     print(type(process))
243
244                     if process_pid:
245                         sum_all += process_cpu_percent
246
247                     if process_pid == script_pid:
248                         cpu_percent += process_cpu_percent
249             except (psutil.NoSuchProcess, psutil.AccessDenied,
250 psutil.ZombieProcess):
251                 pass
252
253         if sum_all != 0:
254             return cpu_percent/sum_all
255         else:
256             return 0.0

```

`get_manufacturer` property

The hardware manufacturer.

Returns:

Name	Type	Description
CpuType	CpuType	A type of CPU.


Example


```

from monitor import Monitor

monitor = Monitor({'cpu': True})

components = monitor.get_monitored_components()
print(components['cpu']['component'].get_manufacturer) #CpuType.INTEL

```

`__get_power_on_linux()`

Returns the value of the CPU power in W in Linux.

Returns:

Name	Type	Description
float	float	CPU power.

Source code in power_pyro\cpu.py

```
159 def __get_power_on_linux(self) -> float:
160     """ Returns the value of the CPU power in W in Linux.
161
162     Returns:
163         float: CPU power.
164     """
165
166     try:
167         command = ["sudo", "perf", "stat", "-e", "power/energy-pkg/",
168 "sleep", "0.1"]
169         power = subprocess.run(command, capture_output=True, text=True)
170
171         power = power.stderr.split(" ")
172         power = [string for string in power if string.strip()]
173
174         for index, string in enumerate(power):
175             if string.find('\n\n') != -1:
176                 power = power[index + 1]
177                 power = power.replace(",", ", ")
178                 break
179
180         power = float(power)/0.1
181     except (PermissionError, subprocess.SubprocessError, AttributeError,
182 IndexError, ValueError) as e:
183         print('Error getting power from CPU: ', str(e))
184
185     return power
```

__get_power_on_windows()

Returns the value of the CPU power in W in Windows.

Returns:

Name	Type	Description
float	float	CPU power.

Source code in power_pyro\cpu.py

```
185 def __get_power_on_windows(self) -> float:
186     """ Returns the value of the CPU power in W in Windows.
187
188     Returns:
189         float: CPU power.
190     """
191     try:
192         cpu = next((hardware for hardware in self.computer.Hardware if
193 hardware.HardwareType == HardwareType.Cpu), None)
194         cpu.Update()
195         time.sleep(0.1)
196
197         power = next((sensor for sensor in cpu.Sensors if
198 sensor.SensorType == SensorType.Power and (sensor.Name == "CPU Package"
199 or sensor.Name == "Package")))
200         power = power.Value
201     except AttributeError as e:
202         print('Error getting power from CPU: ', str(e))
203
204     return power
```

__update_hardware_name_linux()

Set the CPU name.

Raises:

Type	Description
HardwareNameIdentifyException	Unable to identify CPU name in Linux.

Source code in power_pyro\cpu.py

```
140 def __update_hardware_name_linux(self) -> None:
141     """ Set the CPU name.
142
143     Raises:
144         HardwareNameIdentifyException: Unable to identify CPU name in
145 Linux.
146     """
147
148     try:
149         self.set_name = cpuinfo.get_cpu_info()['brand_raw']
150     except (ModuleNotFoundError, KeyError):
151         raise HardwareNameIdentifyException(HT.CPU)
```

`__update_hardware_name_windows()`

Set the CPU name.

Raises:

Type	Description
<code>HardwareNameIdentifyException</code>	Unable to identify CPU name in Windows.

” Source code in `power_pyro\cpu.py`

```
126 def __update_hardware_name_windows(self) -> None:
127     """ Set the CPU name.
128
129     Raises:
130         HardwareNameIdentifyException: Unable to identify CPU name in
131     Windows.
132     """
133
134     try:
135         wmi_session = wmi.WMI()
136
137         self.set_name = wmi_session.Win32_Processor()[0].Name
138     except (ModuleNotFoundError, wmi.x_wmi, IndexError, AttributeError):
139         raise HardwareNameIdentifyException(HT.CPU)
```

`__update_manufacture_linux()`

Update hardware manufacturer when running on Linux OS.

Raises:

Type	Description
<code>IdentifyHardwareManufacturerException</code>	If the hardware manufacturer

Source code in `power_pyro\cpu.py`

```
96 def __update_manufacture_linux(self) -> None:
97     """Update hardware manufacturer when running on Linux OS.
98
99     Raises:
100         IdentifyHardwareManufacturerException: If the hardware
101     manufacturer
102         cannot be identified.
103     """
104
105     try:
106         info = cpuinfo.get_cpu_info()
107
108         manufacturer = info['vendor_id_raw']
109
110         if manufacturer == 'GenuineIntel':
111             self.__manufacturer = CpuType.INTEL
112         elif manufacturer == 'AuthenticAMD':
113             self.__manufacturer = CpuType.AMD
114         else:
115             raise IdentifyHardwareManufacturerException(HT.CPU)
116     except (ModuleNotFoundError, KeyError):
117         raise IdentifyHardwareManufacturerException(HT.CPU)
```

`__update_manufacture_windows()`

Update hardware manufacturer when running on Windows OS.

Raises:

Type	Description
<code>IdentifyHardwareManufacturerException</code>	If the hardware manufacturer

Source code in `power_pyro\cpu.py`

```
74 def __update_manufacture_windows(self) -> None:
75     """Update hardware manufacturer when running on Windows OS.
76
77     Raises:
78         IdentifyHardwareManufacturerException: If the hardware
79     manufacturer
80         cannot be identified.
81     """
82
83     try:
84         wmi_session = wmi.WMI()
85
86         manufacturer = wmi_session.Win32_Processor()[0].Manufacturer
87
88         if manufacturer == 'GenuineIntel':
89             self.__manufacturer = CpuType.INTEL
90         elif manufacturer == 'AuthenticAMD':
91             self.__manufacturer = CpuType.AMD
92         else:
93             raise IdentifyHardwareManufacturerException(HT.CPU)
94     except (ModuleNotFoundError, wmi.X_WMI, IndexError, AttributeError):
95         raise IdentifyHardwareManufacturerException(HT.CPU)
```

`get_cpu_percent_for_process()`

Returns the percentage value of the monitored process on the CPU.

Returns:

Name	Type	Description
<code>float</code>	<code>float</code>	CPU percent.

Example

```
from monitor import Monitor

monitor = Monitor({'cpu': True})
components = monitor.get_monitored_components()
print(components['cpu']['component'].get_cpu_percent_for_process()) # 0.37
```

Source code in `power_pyro\cpu.py`

```
203 def get_cpu_percent_for_process(self) -> float:
204     """ Returns the percentage value of the monitored process on the CPU.
205
206     Returns:
207         float: CPU percent.
208
209     Example:
210         ```python
211
212         from monitor import Monitor
213
214         monitor = Monitor({'cpu': True})
215         components = monitor.get_monitored_components()
216         print(components['cpu']
217 ['component'].get_cpu_percent_for_process()) # 0.37
218         ```
219     """
220     script_pid = os.getpid()
221     sum_all = 0
222     cpu_percent = 0
223     for process in psutil.process_iter():
224         try:
225             with process.oneshot():
226                 process_pid = process.pid
227                 process_cpu_percent = process.cpu_percent()
228                 print(type(process))
229
230                 if process_pid:
231                     sum_all += process_cpu_percent
232
233                     if process_pid == script_pid:
234                         cpu_percent += process_cpu_percent
235         except (psutil.NoSuchProcess, psutil.AccessDenied,
236 psutil.ZombieProcess):
237             pass
238
239     if sum_all != 0:
240         return cpu_percent/sum_all
241     else:
242         return 0.0
```

Módulo GPU

Gpu

Bases: `ProcessingUnit`

Represents a Graphics Processing Unit (GPU) responsible for accessing and retrieving power consumption values from hardware sensors.

Attributes:

Name	Type	Description
<code>__manufacturer</code>	<code>GpuType</code>	GPU type.

```

21 class Gpu(ProcessingUnit):
22     """Represents a Graphics Processing Unit (GPU) responsible
23         for accessing and retrieving power consumption values
24         from hardware sensors.
25
26     Attributes:
27         __manufacturer (GpuType): GPU type.
28     """
29
30     def __init__(self, operating_system: OsType):
31         super().__init__(operating_system)
32         self.__manufacturer: GpuType
33
34         if operating_system == OsType.WINDOWS:
35             self.computer.IsGpuEnabled = True
36
37         self._update_manufacture()
38         self._update_hardware_name()
39
40     @property
41     def get_manufacturer(self) -> GpuType:
42         """ The hardware manufacturer.
43
44     Returns:
45         GpuType: A type of GPU.
46
47     Example:
48         ```python
49
50         from monitor import Monitor
51
52         monitor = Monitor({'gpu': True})
53
54         components = monitor.get_monitored_components()
55         print(components['gpu']['component'].get_manufacturer)
56 #GpuType.NVIDIA
57
58         ...
59
60         """
61         return self.__manufacturer
62
63     def __is_there_dedicated_gpu_windows(self) -> bool:
64         """ Check if it is a dedicated gpu in Windows OS.
65
66     Returns:
67         bool:
68             - 'True' if a dedicated gpu is found on Windows.
69             - 'False' if a dedicated gpu is not found in Windows.
70         """
71         computer = Computer()
72         computer.Open()
73         computer.IsGpuEnabled = True
74
75         gpu = next((hardware for hardware in computer.Hardware if
76                     (hardware.HardwareType == HardwareType.GpuIntel or
77                     hardware.HardwareType == HardwareType.GpuAmd or

```

```

78
79 hardware.HardwareType == HardwareType.GpuNvidia)), None)
80
81     if gpu != None:
82         gpu.Update()
83         time.sleep(0.1)
84
85         if next((sensor for sensor in gpu.Sensors if sensor.Name ==
86 "D3D Dedicated Memory Used"), None) != None:
87             computer.Close()
88             return True
89
90     computer.Close()
91     return False
92
93     def _update_manufacture(self) -> None:
94         if self.operating_system == OsType.WINDOWS:
95             self.__update_manufacture_windows()
96
97         elif self.operating_system == OsType.LINUX:
98             self.__update_manufacture_linux()
99         else:
100             raise OSError("Unable to identify operating system")
101
102     def __update_manufacture_windows(self) -> None:
103         """Update hardware manufacturer when running on Windows OS.
104
105         Raises:
106             IdentifyHardwareManufacturerException: If the hardware
107 manufacturer
108             cannot be identified.
109
110             ResourceUnavailableException: If a dedicated GPU is not found
111 in Windows.
112         """
113         if not self.__is_there_dedicated_gpu_windows():
114             raise ResourceUnavailableException("GPU", "Resource not
115 found!")
116
117         computer = Computer()
118         computer.Open()
119         computer.IsGpuEnabled = True
120
121         for hardware in computer.Hardware:
122             hardware_type = str(hardware.HardwareType)
123
124             if hardware_type == 'GpuNvidia':
125                 self.__manufacturer = GpuType.NVIDIA
126             elif hardware_type == 'GpuAmd':
127                 self.__manufacturer = GpuType.AMD
128             else:
129                 raise IdentifyHardwareManufacturerException(HT.GPU)
130
131         computer.Close()
132
133     def __update_manufacture_linux(self) -> None:
134         """Update hardware manufacturer when running on Linux OS.
135
136         Raises:
137             IdentifyHardwareManufacturerException: If the hardware
138 manufacturer

```

```

139         cannot be identified.
140         """
141
142         if self.__is_there_nvidia_on_linux():
143             self.__manufacturer = GpuType.NVIDIA
144         elif self.__is_there_amd_on_linux():
145             self.__manufacturer = GpuType.AMD
146         else:
147             raise IdentifyHardwareManufacturerException(HT.GPU)
148
149     def __update_hardware_name(self) -> None:
150         if self.operating_system == OsType.WINDOWS:
151             self.__update_hardware_name_windows()
152         elif self.operating_system == OsType.LINUX:
153             self.__update_hardware_name_linux
154         else:
155             raise OSError("Unable to identify operating system")
156
157     def __update_hardware_name_windows(self) -> None:
158         """ Set the GPU name.
159
160         Raises:
161             ResourceUnavailableException: If a dedicated GPU is not found
162             in Windows.
163         """
164         if not self.__is_there_dedicated_gpu_windows():
165             raise ResourceUnavailableException("GPU", "Resource not
166             found!")
167
168         computer = Computer()
169         computer.Open()
170         computer.IsGpuEnabled = True
171
172         for hardware in computer.Hardware:
173             self.set_name = hardware.Name
174
175         computer.Close()
176
177     def __update_hardware_name_linux(self) -> None:
178         """ Set the GPU name.
179
180         Raises:
181             HardwareNameIdentifyException: Unable to identify GPU name in
182             Linux.
183         """
184         try:
185             if self.__is_there_nvidia_on_linux():
186                 self.set_name = subprocess.check_output("nvidia-smi --
187                 query-gpu=name --format=csv,noheader", shell=True).decode().strip()
188             elif self.__is_there_amd_on_linux():
189                 output = subprocess.check_output("lspci | grep -i vga",
190                 shell=True).decode().strip()
191                 self.set_name = re.findall(r'\w+ \w+ \w+ \w+ / \w+
192                 \w+\W\w+', output)
193             else:
194                 raise HardwareNameIdentifyException(HT.GPU)
195             except (FileNotFoundError, subprocess.CalledProcessError,
196             UnicodeDecodeError):
197                 raise HardwareNameIdentifyException(HT.GPU)
198
199     def get_power(self) -> float:

```

```

200         if self.operating_system == OsType.WINDOWS:
201             return self.__get_power_on_windows()
202
203         else:
204             if self.__manufacturer == GpuType.NVIDIA:
205                 return self.__get_nvidia_power_on_linux()
206             elif self.__manufacturer == GpuType.AMD:
207                 return self.__get_amd_power_on_linux()
208
209     def __get_power_on_windows(self) -> float:
210         """ Returns the value of the GPU power in W in Windows.
211
212         Returns:
213             float: GPU power.
214         """
215         gpu = next((hardware for hardware in self.computer.Hardware if
216 (hardware.HardwareType == HardwareType.GpuIntel or
217 hardware.HardwareType == HardwareType.GpuAmd or
218 hardware.HardwareType == HardwareType.GpuNvidia)), None)
219         gpu.Update()
220         time.sleep(0.1)
221
222         power = next((sensor for sensor in gpu.Sensors if
223 sensor.SensorType == SensorType.Power and (sensor.Name == "GPU Power" or
224 sensor.Name == "GPU Package")))
225         return power.Value
226
227     def __is_there_nvidia_on_linux(self) -> bool:
228         """ Check if the GPU present in linux is NVIDIA.
229
230         Returns:
231             bool:
232                 - 'True' if you have NVIDIA GPU on linux.
233                 - 'False' if you don't have NVIDIA GPU on linux.
234         """
235         try:
236             subprocess.run(['nvidia-smi'], stdout=subprocess.PIPE,
237 stderr= subprocess.PIPE, check=True)
238             return True
239         except (FileNotFoundError, subprocess.CalledProcessError):
240             return False
241
242     def __get_nvidia_power_on_linux(self) -> float:
243         """ Returns the value of the NVIDIA GPU power in W in Linux.
244
245         Returns:
246             float: GPU power.
247         """
248         try:
249             result = subprocess.run(["nvidia-smi", "--query-
250 gpu=power.draw", "--format=csv,noheader,nounits"],
251                                     stdout=subprocess.PIPE,
252                                     stderr=subprocess.PIPE,
253                                     text=True,
254                                     check=True)
255
256             return float(result.stdout.strip())
257         except Exception as e:
258             print('Error getting power from GPU: ', str(e))
259
260

```

```

261         return 0.0
262
263     def __is_there_amd_on_linux(self) -> bool:
264         """ Check if the GPU present in Linux is AMD.
265
266         Returns:
267             bool:
268                 - 'True' if you have AMD GPU on Linux.
269                 - 'False' if you don't have AMD GPU on Linux.
270
271         Raises:
272             Exception: Error when checking AMD dedicated video card on
273 Linux.
274         """
275         try:
276             result = subprocess.check_output(['lspci', '-nnk'],
277 universal_newlines=True)
278
279             for line in result.splitlines():
280                 if re.search(r"( VGA | 3D )", line) and re.search(r"AMD",
281 line.upper()):
282                     return True
283
284             return False
285         except Exception as e:
286             raise Exception(f'Error checking for AMD graphics card:{e}')
287
288     def __get_amd_power_on_linux(self) -> float:
289         """ Returns the value of the AMD GPU power in W in Linux.
290
291         Returns:
292             float: GPU power.
293         """
294         try:
295             hwmon_path = '/sys/class/hwmon/'
296
297             for hwmon in os.listdir(hwmon_path):
298                 hwmon_dir = os.path.join(hwmon_path, hwmon)
299
300                 name_file = os.path.join(hwmon_dir, 'name')
301                 if os.path.exists(name_file):
302                     with open(name_file, 'r') as file:
303                         device = file.read().strip()
304
305                     if device == 'amdgpu':
306                         power_file = os.path.join(hwmon_dir,
307 'power1_input')
308
309                         if os.path.exists(power_file):
310                             with open(power_file, 'r') as file:
311                                 power = float(file.read().strip())
312
313                                 power /= 10**6
314                                 return power
315         except (FileNotFoundError, PermissionError, ValueError) as e:
316             print('Error getting power from GPU: ', str(e))

```

get_manufacturer property

The hardware manufacturer.

Returns:

Name	Type	Description
GpuType	GpuType	A type of GPU.



Example



```
from monitor import Monitor

monitor = Monitor({'gpu': True})

components = monitor.get_monitored_components()
print(components['gpu']['component'].get_manufacturer) #GpuType.NVIDIA
```

`__get_amd_power_on_linux()`

Returns the value of the AMD GPU power in W in Linux.

Returns:

Name	Type	Description
float	float	GPU power.

Source code in power_pyro\gpu.py

```
262 def __get_amd_power_on_linux(self) -> float:
263     """ Returns the value of the AMD GPU power in W in Linux.
264
265     Returns:
266         float: GPU power.
267     """
268     try:
269         hwmon_path = '/sys/class/hwmon/'
270
271         for hwmon in os.listdir(hwmon_path):
272             hwmon_dir = os.path.join(hwmon_path, hwmon)
273
274             name_file = os.path.join(hwmon_dir, 'name')
275             if os.path.exists(name_file):
276                 with open(name_file, 'r') as file:
277                     device = file.read().strip()
278
279                 if device == 'amdgpu':
280                     power_file = os.path.join(hwmon_dir, 'power1_input')
281                     if os.path.exists(power_file):
282                         with open(power_file, 'r') as file:
283                             power = float(file.read().strip())
284
285                             power /= 10**6
286                             return power
287     except (FileNotFoundError, PermissionError, ValueError) as e:
288         print('Error getting power from GPU: ', str(e))
```

__get_nvidia_power_on_linux()

Returns the value of the NVIDIA GPU power in W in Linux.

Returns:

Name	Type	Description
float	float	GPU power.

Source code in `power_pyro\gpu.py`

```
222 def __get_nvidia_power_on_linux(self) -> float:
223     """ Returns the value of the NVIDIA GPU power in W in Linux.
224
225     Returns:
226         float: GPU power.
227     """
228     try:
229         result = subprocess.run(["nvidia-smi", "--query-gpu=power.draw",
230                                "--format=csv,noheader,nounits"],
231                                stdout=subprocess.PIPE,
232                                stderr=subprocess.PIPE,
233                                text=True,
234                                check=True)
235
236         return float(result.stdout.strip())
237     except Exception as e:
238         print('Error getting power from GPU: ', str(e))
239         return 0.0
```

`__get_power_on_windows()`

Returns the value of the GPU power in W in Windows.

Returns:

Name	Type	Description
<code>float</code>	<code>float</code>	GPU power.

Source code in power_pyro\gpu.py

```
193 def __get_power_on_windows(self) -> float:
194     """ Returns the value of the GPU power in W in Windows.
195
196     Returns:
197         float: GPU power.
198     """
199     gpu = next((hardware for hardware in self.computer.Hardware if
200 (hardware.HardwareType == HardwareType.GpuIntel or
201 hardware.HardwareType == HardwareType.GpuAmd or
202 hardware.HardwareType == HardwareType.GpuNvidia)), None)
203     gpu.Update()
204     time.sleep(0.1)
205
206     power = next((sensor for sensor in gpu.Sensors if sensor.SensorType
207 == SensorType.Power and (sensor.Name == "GPU Power" or sensor.Name ==
208 "GPU Package"))))
209     return power.Value
```

__is_there_amd_on_linux()

Check if the GPU present in Linux is AMD.

Returns:

Name	Type	Description
bool	bool	<ul style="list-style-type: none">'True' if you have AMD GPU on Linux.'False' if you don't have AMD GPU on Linux.

Raises:

Type	Description
Exception	Error when checking AMD dedicated video card on Linux.

Source code in power_pyro\gpu.py

```
240 def __is_there_amd_on_linux(self) -> bool:
241     """ Check if the GPU present in Linux is AMD.
242
243     Returns:
244         bool:
245             - 'True' if you have AMD GPU on Linux.
246             - 'False' if you don't have AMD GPU on Linux.
247
248     Raises:
249         Exception: Error when checking AMD dedicated video card on Linux.
250     """
251     try:
252         result = subprocess.check_output(['lspci', '-nnk'],
253 universal_newlines=True)
254
255         for line in result.splitlines():
256             if re.search(r"( VGA | 3D )", line) and re.search(r"AMD",
257 line.upper()):
258                 return True
259
260         return False
261     except Exception as e:
262         raise Exception(f'Error checking for AMD graphics card:{e}')
```

`__is_there_dedicated_gpu_windows()`

Check if it is a dedicated gpu in Windows OS.

Returns:

Name	Type	Description
bool	bool	<ul style="list-style-type: none">'True' if a dedicated gpu is found on Windows.'False' if a dedicated gpu is not found in Windows.

Source code in power_pyro\gpu.py

```
61 def __is_there_dedicated_gpu_windows(self) -> bool:
62     """ Check if it is a dedicated gpu in Windows OS.
63
64     Returns:
65         bool:
66             - 'True' if a dedicated gpu is found on Windows.
67             - 'False' if a dedicated gpu is not found in Windows.
68     """
69     computer = Computer()
70     computer.Open()
71     computer.IsGpuEnabled = True
72
73     gpu = next((hardware for hardware in computer.Hardware if
74 (hardware.HardwareType == HardwareType.GpuIntel or
75 hardware.HardwareType == HardwareType.GpuAmd or
76 hardware.HardwareType == HardwareType.GpuNvidia)), None)
77
78     if gpu != None:
79         gpu.Update()
80         time.sleep(0.1)
81
82         if next((sensor for sensor in gpu.Sensors if sensor.Name == "D3D
83 Dedicated Memory Used"), None) != None:
84             computer.Close()
85             return True
86
87     computer.Close()
88     return False
```

__is_there_nvidia_on_linux()

Check if the GPU present in linux is NVIDIA.

Returns:

Name	Type	Description
bool	bool	<ul style="list-style-type: none">'True' if you have NVIDIA GPU on linux.'False' if you don't have NVIDIA GPU on linux.

Source code in `power_pyro\gpu.py`

```
208 def __is_there_nvidia_on_linux(self) -> bool:
209     """ Check if the GPU present in linux is NVIDIA.
210
211     Returns:
212         bool:
213             - 'True' if you have NVIDIA GPU on linux.
214             - 'False' if you don't have NVIDIA GPU on linux.
215     """
216     try:
217         subprocess.run(['nvidia-smi'], stdout=subprocess.PIPE, stderr=
218         subprocess.PIPE, check=True)
219         return True
220     except (FileNotFoundError, subprocess.CalledProcessError):
221         return False
```

`__update_hardware_name_linux()`

Set the GPU name.

Raises:

Type	Description
<code>HardwareNameIdentifyException</code>	Unable to identify GPU name in Linux.

Source code in power_pyro\gpu.py

```
166 def __update_hardware_name_linux(self) -> None:
167     """ Set the GPU name.
168
169     Raises:
170         HardwareNameIdentifyException: Unable to identify GPU name in
171     Linux.
172     """
173     try:
174         if self.__is_there_nvidia_on_linux():
175             self.set_name = subprocess.check_output("nvidia-smi --query-
176 gpu=name --format=csv,noheader", shell=True).decode().strip()
177         elif self.__is_there_amd_on_linux():
178             output = subprocess.check_output("lspci | grep -i vga",
179 shell=True).decode().strip()
180             self.set_name = re.findall(r'\w+ \w+ \w+ \w+ / \w+ \w+\W\w+',
181 output)
182         else:
183             raise HardwareNameIdentifyException(HT.GPU)
184     except (FileNotFoundError, subprocess.CalledProcessError,
185 UnicodeDecodeError):
186         raise HardwareNameIdentifyException(HT.GPU)
```

__update_hardware_name_windows()

Set the GPU name.

Raises:

Type	Description
ResourceUnavailableException	If a dedicated GPU is not found in Windows.

Source code in `power_pyro\gpu.py`

```
148 def __update_hardware_name_windows(self) -> None:
149     """ Set the GPU name.
150
151     Raises:
152         ResourceUnavailableException: If a dedicated GPU is not found in
153     Windows.
154     """
155     if not self.__is_there_dedicated_gpu_windows():
156         raise ResourceUnavailableException("GPU", "Resource not found!")
157
158     computer = Computer()
159     computer.Open()
160     computer.IsGpuEnabled = True
161
162     for hardware in computer.Hardware:
163         self.set_name = hardware.Name
164
165     computer.Close()
```

`__update_manufacture_linux()`

Update hardware manufacturer when running on Linux OS.

Raises:

Type	Description
<code>IdentifyHardwareManufacturerException</code>	If the hardware manufacturer

Source code in `power_pyro\gpu.py`

```
125 def __update_manufacture_linux(self) -> None:
126     """Update hardware manufacturer when running on Linux OS.
127
128     Raises:
129         IdentifyHardwareManufacturerException: If the hardware
130     manufacturer
131         cannot be identified.
132     """
133
134     if self.__is_there_nvidia_on_linux():
135         self.__manufacturer = GpuType.NVIDIA
136     elif self.__is_there_amd_on_linux():
137         self.__manufacturer = GpuType.AMD
138     else:
139         raise IdentifyHardwareManufacturerException(HT.GPU)
```

`__update_manufacture_windows()`

Update hardware manufacturer when running on Windows OS.

Raises:

Type	Description
<code>IdentifyHardwareManufacturerException</code>	If the hardware manufacturer
<code>ResourceUnavailableException</code>	If a dedicated GPU is not found in Windows.

Source code in `power_pyro\gpu.py`

```
97 def __update_manufacture_windows(self) -> None:
98     """Update hardware manufacturer when running on Windows OS.
99
100     Raises:
101         IdentifyHardwareManufacturerException: If the hardware
102     manufacturer
103         cannot be identified.
104
105         ResourceUnavailableException: If a dedicated GPU is not found in
106     Windows.
107     """
108     if not self.__is_there_dedicated_gpu_windows():
109         raise ResourceUnavailableException("GPU", "Resource not found!")
110
111     computer = Computer()
112     computer.Open()
113     computer.IsGpuEnabled = True
114
115     for hardware in computer.Hardware:
116         hardware_type = str(hardware.HardwareType)
117
118         if hardware_type == 'GpuNvidia':
119             self.__manufacturer = GpuType.NVIDIA
120         elif hardware_type == 'GpuAmd':
121             self.__manufacturer = GpuType.AMD
122         else:
123             raise IdentifyHardwareManufacturerException(HT.GPU)
124
125     computer.Close()
```

Módulo Memória

Memory

Bases: `HardwareComponent`

Represents a Memory component responsible for calculating power consumption based on the amount of memory used.

Attributes:

Name	Type	Description
<code>__WATT_PER_GB</code>	<code>float</code>	Power consumption per GB of memory.

```

12 class Memory(HardwareComponent):
13     """Represents a Memory component responsible for calculating power
14     consumption
15     based on the amount of memory used.
16
17     Attributes:
18         __WATT_PER_GB (float): Power consumption per GB of memory.
19     """
20     def __init__(self, operating_system: OsType):
21         super().__init__(operating_system)
22         self.__WATT_PER_GB: float = self.__watt_per_gb()
23
24     def __watt_per_gb(self) -> float:
25         """Calculates the power consumption per GB of memory.
26
27         Returns:
28             float: Power consumption per GB of memory.
29
30         Raises:
31             OSError: If the operating system is not recognized.
32         """
33         if self.operating_system == OsType.WINDOWS:
34             return self.__watt_per_gb_on_windows()
35         elif self.operating_system == OsType.LINUX:
36             return self.__watt_per_gb_on_linux()
37         else:
38             raise OSError("Unable to identify operating system")
39
40     def __watt_per_gb_on_windows(self) -> float:
41         """Calculates the power consumption per GB of memory on Windows
42         OS.
43
44         Returns:
45             float: Power consumption per GB.
46
47         Raises:
48             RuntimeError: Unable to get information from memory.
49         """
50         try:
51             BYTES_TO_GIGABYTES = 1024**3
52             wmi_session = wmi.WMI()
53
54             num_memory_modules = len(wmi_session.Win32_PhysicalMemory())
55
56             memory_module = wmi_session.Win32_PhysicalMemory()[0]
57             gb_per_module =
58             int(memory_module.Capacity)/BYTES_TO_GIGABYTES
59             except (ModuleNotFoundError, IndexError, TypeError, wmi.x_wmi):
60                 raise RuntimeError("Unable to get watts per GB information
61                 from memory")
62
63             return (5 * num_memory_modules)/gb_per_module
64
65     def __watt_per_gb_on_linux(self) -> float:
66         """Calculates the power consumption per GB of memory on Linux OS.
67
68         Returns:

```

```

69         float: Power consumption per GB.
70
71     Raises:
72         RuntimeError: Unable to get information from memory.
73     """
74     try:
75         output = subprocess.check_output(["sudo", "dmidecode", "-t",
76 "memory"], universal_newlines=True)
77
78         num_memory_modules_found = len(re.findall(r"\tSize:",
79 output))
80         number_unused_memory_modules = len(re.findall(r"Size: No
81 Module Installed", output))
82
83         num_memory_modules = num_memory_modules_found -
84 number_unused_memory_modules
85
86         gb_per_module = re.findall(r"\tSize: \d+ \w+", output)
87         gb_per_module = gb_per_module[0].split(": ")
88         gb_per_module = gb_per_module[1].split(" ")
89         gb_per_module = int(gb_per_module[0])
90     except (FileNotFoundError, PermissionError,
91 subprocess.CalledProcessError, IndexError, ValueError):
92         raise RuntimeError("Unable to get watts per GB information
93 from memory")
94
95     return (5 * num_memory_modules)/gb_per_module
96
97     def get_power(self) -> float:
98         """Returns the power consumption of the memory in W.
99
100     Returns:
101         float: Memory power consumption.
102
103     Example:
104         Monitor and print memory power consumption:
105
106         ```python
107         from power_pyro import Monitor
108
109         monitor = Monitor({'memory': True}) # Enable memory
110     monitoring
111         power = monitor.memory.get_power() # Access memory subsystem
112         print(f"Current memory power: {power:.2f} W") # 15.2 W
113         ```
114
115     """
116     try:
117         pid = os.getpid()
118         process = psutil.Process(pid)
119
120         power = process.memory_info().rss
121         power /= (1024 ** 3)
122         power *= self.__WATT_PER_GB
123     except (psutil.NoSuchProcess, psutil.AccessDenied,
124 psutil.ZombieProcess) as e:
125         print('Error getting power from memory:', str(e))
126
127     return power

```

`__watt_per_gb()`

Calculates the power consumption per GB of memory.

Returns:

Name	Type	Description
float	float	Power consumption per GB of memory.

Raises:

Type	Description
OSError	If the operating system is not recognized.

Source code in power_pyro\memory.py

```
23 def __watt_per_gb(self) -> float:
24     """Calculates the power consumption per GB of memory.
25
26     Returns:
27         float: Power consumption per GB of memory.
28
29     Raises:
30         OSErrror: If the operating system is not recognized.
31     """
32     if self.operating_system == OsType.WINDOWS:
33         return self.__watt_per_gb_on_windows()
34     elif self.operating_system == OsType.LINUX:
35         return self.__watt_per_gb_on_linux()
36     else:
37         raise OSErrror("Unable to identify operating system")
```

`__watt_per_gb_on_linux()`

Calculates the power consumption per GB of memory on Linux OS.

Returns:

Name	Type	Description
float	float	Power consumption per GB.

Raises:

Type	Description
<code>RuntimeError</code>	Unable to get information from memory.

Source code in `power_pyro\memory.py`

```
61 def __watt_per_gb_on_linux(self) -> float:
62     """Calculates the power consumption per GB of memory on Linux OS.
63
64     Returns:
65         float: Power consumption per GB.
66
67     Raises:
68         RuntimeError: Unable to get information from memory.
69     """
70     try:
71         output = subprocess.check_output(["sudo", "dmidecode", "-t",
72     "memory"], universal_newlines=True)
73
74         num_memory_modules_found = len(re.findall(r"\tSize:", output))
75         number_unused_memory_modules = len(re.findall(r"Size: No Module
76 Installed", output))
77
78         num_memory_modules = num_memory_modules_found -
79 number_unused_memory_modules
80
81         gb_per_module = re.findall(r"\tSize: \d+ \w+", output)
82         gb_per_module = gb_per_module[0].split(": ")
83         gb_per_module = gb_per_module[1].split(" ")
84         gb_per_module = int(gb_per_module[0])
85     except (FileNotFoundError, PermissionError,
subprocess.CalledProcessError, IndexError, ValueError):
        raise RuntimeError("Unable to get watts per GB information from
memory")
        return (5 * num_memory_modules)/gb_per_module
```

`__watt_per_gb_on_windows()`

Calculates the power consumption per GB of memory on Windows OS.

Returns:

Name	Type	Description
<code>float</code>	<code>float</code>	Power consumption per GB.

Raises:

Type	Description
<code>RuntimeError</code>	Unable to get information from memory.

Source code in `power_pyro\memory.py`

```
39 def __watt_per_gb_on_windows(self) -> float:
40     """Calculates the power consumption per GB of memory on Windows OS.
41
42     Returns:
43         float: Power consumption per GB.
44
45     Raises:
46         RuntimeError: Unable to get information from memory.
47     """
48     try:
49         BYTES_TO_GIGABYTES = 1024**3
50         wmi_session = wmi.WMI()
51
52         num_memory_modules = len(wmi_session.Win32_PhysicalMemory())
53
54         memory_module = wmi_session.Win32_PhysicalMemory()[0]
55         gb_per_module = int(memory_module.Capacity)/BYTES_TO_GIGABYTES
56     except (ModuleNotFoundError, IndexError, TypeError, wmi.X_WMI):
57         raise RuntimeError("Unable to get watts per GB information from
58 memory")
59
60     return (5 * num_memory_modules)/gb_per_module
```

`get_power()`

Returns the power consumption of the memory in W.

Returns:

Name	Type	Description
<code>float</code>	<code>float</code>	Memory power consumption.

Example

Monitor and print memory power consumption:

```
from power_pyro import Monitor

monitor = Monitor({'memory': True}) # Enable memory monitoring
power = monitor.memory.get_power() # Access memory subsystem
print(f"Current memory power: {power:.2f} W") # 15.2 W
```

Source code in `power_pyro\memory.py`

```
87 def get_power(self) -> float:
88     """Returns the power consumption of the memory in W.
89
90     Returns:
91         float: Memory power consumption.
92
93     Example:
94         Monitor and print memory power consumption:
95
96         ```python
97         from power_pyro import Monitor
98
99         monitor = Monitor({'memory': True}) # Enable memory monitoring
100         power = monitor.memory.get_power() # Access memory subsystem
101         print(f"Current memory power: {power:.2f} W") # 15.2 W
102         ```
103
104     """
105     try:
106         pid = os.getpid()
107         process = psutil.Process(pid)
108
109         power = process.memory_info().rss
110         power /= (1024 ** 3)
111         power *= self.__WATT_PER_GB
112     except (psutil.NoSuchProcess, psutil.AccessDenied,
113             psutil.ZombieProcess) as e:
114         print('Error getting power from memory:', str(e))
115
116     return power
```