

AMK install

06/06/2024

Installing and Using the Basic Advanced Memory Kernel (AMK)

Below, you will find a step-by-step guide to installing and using the basic Advanced Memory Kernel (AMK) library. This guide includes setting up your environment, installing dependencies, and running an example script.

Step-by-Step Guide

Step 1: Setting Up Your Environment

First, make sure you have Python installed on your system. It's recommended to use a virtual environment to manage your dependencies.

1. **Create a Virtual Environment**

```
```bash
python -m venv amk_env
```
```

2. **Activate the Virtual Environment**

On Windows:

```
```bash
amk_env\Scripts\activate
```
```

On macOS and Linux:

```
```bash
source amk_env/bin/activate
```
```

Step 2: Creating the Project Structure

Create the necessary folders and files for your project.

```
```bash
mkdir AdvancedMemoryKernel
cd AdvancedMemoryKernel
mkdir src tests examples docs
touch src/amk.py README.md LICENSE .gitignore setup.py requirements.txt
```
```

Step 3: Writing the AMK Code

Create the `amk.py` file inside the `src` directory and add the following code:

```
```python
from collections import OrderedDict
import tensorflow as tf

class AMKMemoryManager:
 def __init__(self, cache_size=1024):
 self.cache_size = cache_size
 self.cache = OrderedDict()
 self.memory = {}

 def allocate(self, key, value):
 if key in self.cache:
 self.cache.move_to_end(key)
 else:
 if len(self.cache) >= self.cache_size:
 self.cache.popitem(last=False)
 self.cache[key] = value
 self.memory[key] = value
```
```

```

def retrieve(self, key):
    if key in self.cache:
        self.cache.move_to_end(key)
        return self.cache[key]
    return self.memory.get(key)

def free(self, key):
    if key in self.cache:
        del self.cache[key]
    if key in self.memory:
        del self.memory[key]

def create_tensor(bit_size):
    if bit_size == 16:
        dtype = tf.float16
    elif bit_size == 32:
        dtype = tf.float32
    elif bit_size == 64:
        dtype = tf.float64
    else:
        raise ValueError("Unsupported bit size")

    return tf.Variable(tf.random.uniform([1024, 1024], dtype=dtype))

# Example usage
if __name__ == "__main__":
    amk_manager = AMKMemoryManager()

    tensor_key = 'layer_1_weights'
    tensor_value = tf.random.uniform((1024, 1024))
    amk_manager.allocate(tensor_key, tensor_value)

    retrieved_tensor = amk_manager.retrieve(tensor_key)
    print(retrieved_tensor)

    amk_manager.free(tensor_key)

```

Step 4: Writing the Setup Script

Create the `setup.py` file to make it easy to install the AMK library.

```

python
from setuptools import setup, find_packages

setup(
    name='AdvancedMemoryKernel',
    version='0.1.0',
    packages=find_packages(where="src"),
    package_dir={"": "src"},
    install_requires=[
        'tensorflow>=2.0.0',
    ],
    description='An advanced memory management library for efficient and dynamic memory allocation',
    author='Your Name',
    author_email='your.email@example.com',
    url='https://github.com/your-username/AdvancedMemoryKernel',
    classifiers=[
        'Programming Language :: Python :: 3',
        'License :: OSI Approved :: MIT License',
        'Operating System :: OS Independent',
    ],
)

```

Step 5: Writing the Requirements File

Create the `requirements.txt` file with the necessary dependencies.

```
``text
tensorflow>=2.0.0
```
```

#### Step 6: Adding Example Usage

Create an example script in the `examples` directory.

```
```python
# examples/example_usage.py
from src.amk import AMKMemoryManager, create_tensor

# Initialize AMK Memory Manager
amk_manager = AMKMemoryManager()

# Allocate a tensor
tensor_key = 'layer_1_weights'
tensor_value = tf.random.uniform((1024, 1024))
amk_manager.allocate(tensor_key, tensor_value)

# Retrieve the tensor
retrieved_tensor = amk_manager.retrieve(tensor_key)
print(retrieved_tensor)

# Free the tensor
amk_manager.free(tensor_key)
```
```

#### Step 7: Creating the README File

Provide instructions and documentation in the `README.md` file.

```
```markdown
# Advanced Memory Kernel (AMK)

The Advanced Memory Kernel (AMK) is a state-of-the-art memory management library designed to optimize memory usage and access patterns through dynamic allocation, efficient caching, and support for various bit sizes.

## Features
- Dynamic memory allocation
- Efficient caching using LRU (Least Recently Used)
- Support for various bit sizes (16-bit, 32-bit, 64-bit, etc.)
- Integration with TensorFlow

## Installation
Clone the repository and install the required packages:
```bash
git clone https://github.com/your-username/AdvancedMemoryKernel.git
cd AdvancedMemoryKernel
pip install -r requirements.txt
```

## Usage
Example usage of AMK with TensorFlow:
```python
from src.amk import AMKMemoryManager, create_tensor

Initialize AMK Memory Manager
amk_manager = AMKMemoryManager()

Allocate a tensor
tensor_key = 'layer_1_weights'
tensor_value = tf.random.uniform((1024, 1024))
amk_manager.allocate(tensor_key, tensor_value)

Retrieve the tensor
retrieved_tensor = amk_manager.retrieve(tensor_key)
print(retrieved_tensor)
```
```

```
# Free the tensor
amk_manager.free(tensor_key)
'''
```

```
## Contributing
Contributions are welcome! Please read the [contributing guidelines](docs/CONTRIBUTING.md) first.
```

```
## License
This project is licensed under the MIT License - see the [LICENSE](LICENSE) file for details.
'''
```

Step 8: Installing the Library

Navigate to the project directory and install the library.

```
'''bash
pip install -e .
'''
```

Step 9: Running the Example Script

Run the example script to see AMK in action.

```
'''bash
python examples/example_usage.py
'''
```

Conclusion

This guide provides a comprehensive overview of setting up, installing, and using the Advanced Memory Kernel (AMK). By following these steps, you can leverage AMK's advanced memory management capabilities in your TensorFlow projects and beyond.