Program Listing for File tensor.hpp

Return to documentation for file (include/holoscan/core/domain/tensor.hpp)

/* * SPDX-FileCopyrightText: Copyright (c) 2022-2024 NVIDIA CORPORATION & AFFILIATES. All rights reserved. * SPDX-License-Identifier: Apache-2.0 * * Licensed under the Apache License, Version 2.0 (the "License"); * you may not use this file except in compliance with the License. * You may obtain a copy of the License at * * http://www.apache.org/licenses/LICENSE-2.0 * * Unless required by applicable law or agreed to in writing, software * distributed under the License is distributed on an "AS IS" BASIS, * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. * See the License for the specific language governing permissions and * limitations under the License. */ #ifndef HOLOSCAN_CORE_DOMAIN_TENSOR_HPP #define HOLOSCAN_CORE_DOMAIN_TENSOR_HPP #include <dlpack/dlpack.h> #include <cstdint> #include <functional> #include <memory> #include <string> #include <vector> #include <gxf/std/tensor.hpp> namespace holoscan { *// TODO: keep old class name as an alias? // also differs in that DLManagedTensorContext has additional members dl_shape and dl_strides // using DLManagedTensorCtx = nvidia::gxf::DLManagedTensorContext;* using DLManagedTensorContext = nvidia::gxf::DLManagedTensorContext; using DLManagedMemoryBuffer = nvidia::gxf::DLManagedMemoryBuffer; class Tensor { public: Tensor() = default; explicit Tensor(std::shared_ptr<DLManagedTensorContext>& ctx) : dl_ctx_(ctx) {} explicit Tensor(DLManagedTensor* dl_managed_tensor_ptr); virtual ~Tensor() = default; void* data() const { return dl_ctx_->tensor.dl_tensor.data; } DLDevice device() const { return dl_ctx_->tensor.dl_tensor.device; } DLDataType dtype() const { return dl_ctx_->tensor.dl_tensor.dtype; } std::vector<int64_t> shape() const; std::vector<int64_t> strides() const; bool is_contiguous() const; int64_t size() const; int32_t ndim() const { return dl_ctx_->tensor.dl_tensor.ndim; } uint8_t itemsize() const { return (dl_ctx_->tensor.dl_tensor.dtype.bits * dl_ctx_->tensor.dl_tensor.dtype.lanes + 7) / 8; } int64_t nbytes() const { return size() * itemsize(); } DLManagedTensor* to_dlpack(); std::shared_ptr<DLManagedTensorContext>& dl_ctx() { return dl_ctx_; } protected: std::shared_ptr<DLManagedTensorContext> dl_ctx_; }; DLDevice dldevice_from_pointer(void* ptr); void calc_strides(const DLTensor& tensor, std::vector<int64_t>& strides, bool to_num_elements = false); DLDataType dldatatype_from_typestr(const std::string& typestr); const char* numpy_dtype(const

```
DLDataType dtype); } // namespace holoscan #endif/*
HOLOSCAN_CORE_DOMAIN_TENSOR_HPP */
```