Program Listing for File utils.hpp

Return to documentation for file (modules/holoinfer/src/infer/trt/utils.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 HOLOINFER_TRT_UTILS_H #define HOLOINFER_TRT_UTILS_H #include <cuda_runtime_api.h> #include <algorithm> #include <cassert> #include <cctype> #include <filesystem> #include <fstream> #include <iostream> #include <iterator> #include <memory> #include <numeric> #include <string> #include <vector> #include <NvInfer.h> #include <holoscan/logger/logger.hpp> namespace holoscan { namespace inference { class Logger : public nvinfer1::ILogger { void log(Severity severity, const char* msg) noexcept override { if (severity <= Severity::kWARNING) { try { *// ignore potential fmt::format_error exception* HOLOSCAN_LOG_INFO(msg); } catch (std::exception& e) {} } }; }; struct NetworkOptions { bool use_fp16 = true; std::vector<int32_t> batch_sizes = {1}; int32_t max_batch_size = 1; size_t max_memory = 10000000000; int device_index = 0; }; bool valid_file_path(const std::string& filepath); bool generate_engine_path(const NetworkOptions& options, const std::string& model_path, std::string& engine_name); bool build_engine(const std::string& onnxModelPath, const std::string& engine_name_, const NetworkOptions& network_options_, Logger& logger_); static auto StreamDeleter = [](cudaStream_t* pStream) { if (pStream) { cudaStreamDestroy(*pStream); delete pStream; } }; inline std::unique_ptr<cudaStream_t, decltype(StreamDeleter)> makeCudaStream() { std::unique_ptr<cudaStream_t, decltype(StreamDeleter)> pStream(new cudaStream_t, StreamDeleter); if (cudaStreamCreateWithFlags(pStream.get(), cudaStreamNonBlocking) != cudaSuccess) { pStream.reset(nullptr); } return pStream; } } *// namespace inference* } *// namespace holoscan* #endif*// HOLOINFER_TRT_UTILS_H*