创建 OpenVINO™ Stateful 模型与 Runtime 流水线，赋能 ChatGLM

作者：赵桢、罗成、李亭骞、邹文艺

引言

自大语言模型 (LLM) 成为热点话题以来，涌现了一大批中文大语言模型并在优化平台中得到了积极部署。ChatGLM 正是广受好评的主流中文大语言模型之一。然而，由于 ChatGLM 模型尚未成为 Transformer 生态的原生模型，因此，官方 optimum 扩展库对其仍缺乏支持。本文提供了一种使用 OpenVINO™ opset 重构该模型架构的便捷方法。该方案包含专为 ChatGLM 定制的优化节点，且这些节点都利用英特尔® 高级矩阵扩展（Intel® Advanced Matrix Extensions，缩写为英特尔® AMX）内联和 MHA（Multi-Head Attention，多头注意力）融合实现了高度优化。

*请注意，本文仅介绍了通过为 ChatGLM 创建 OpenVINO™ stateful 模型实现优化的解决方案。本方案受平台限制，必须使用内置了英特尔® AMX 的第四代英特尔® 至强® 可扩展处理器[1]（代号 Sapphire Rapids）。笔者不承诺对该解决方案进行任何维护。

ChatGLM 模型简介

笔者在查看 ChatGLM 原始模型的源码时，发现 ChatGLM 与 Optimum *ModelForCasualML* 并不兼容，而是定义了新的类 *ChatGLMForConditionalGeneration*。该模型的流水线回路包含 3 个主要模块（Embedding、GLMBlock 层和 lm_logits），结构如下：

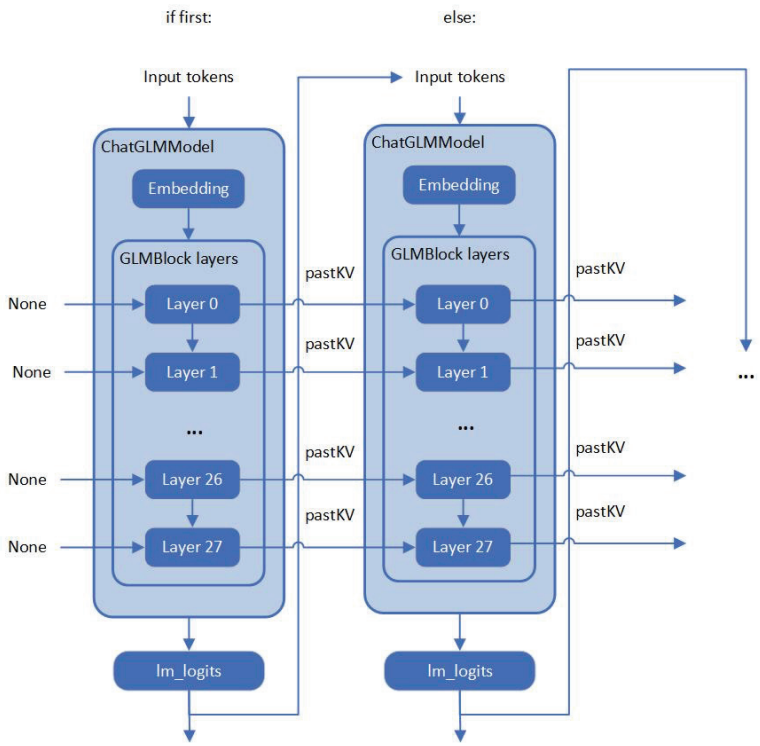

图 1. ChatGLM 的模型结构

如上图所示，整个流水线实际要求模型有两个不同的执行图，使用输入提示符进行首次推理时不需要 KV 缓存作为 *GLMBlock* 层的输入。从第二次迭代开始，QKV 注意力机制的上一次结果将成为当前一轮模型推理的输入。随着生成符的长度不断增加，在流水线推理过程中，模型输入和输出之间将存留大量的大型内存副本。以 ChatGLM6b 默认模型配置“ 为示例，输入和输出阵列之间的内存副本类似于以下伪代码，其内存拷贝的开销由模型的参数 hidden_size 以及迭代的次数决定：

```
while(eos_token_id || max_seq_len){
  memcpy(model_inp, model_outp, num_layer*2*sizeof(model_outp)* hidden_size)
  model_outp.push_back(gen_token)
}
```

1. 因此，本文要解决的两大关键问题是：

- 如何优化模型推理流水线来消除模型输入和输出之间的内存副本
- 如何通过重新设计执行图来优化 *GLMBlock* 模块

构建 OpenVINO™ stateful 模型实现显著优化

首先，需要分析 *GLMBlock* 层的结构，尝试封装一个类并按以下工作流来调用 OpenVINO™ opset。接着，将图形数据序列化为 IR 模型 (.xml, .bin)。

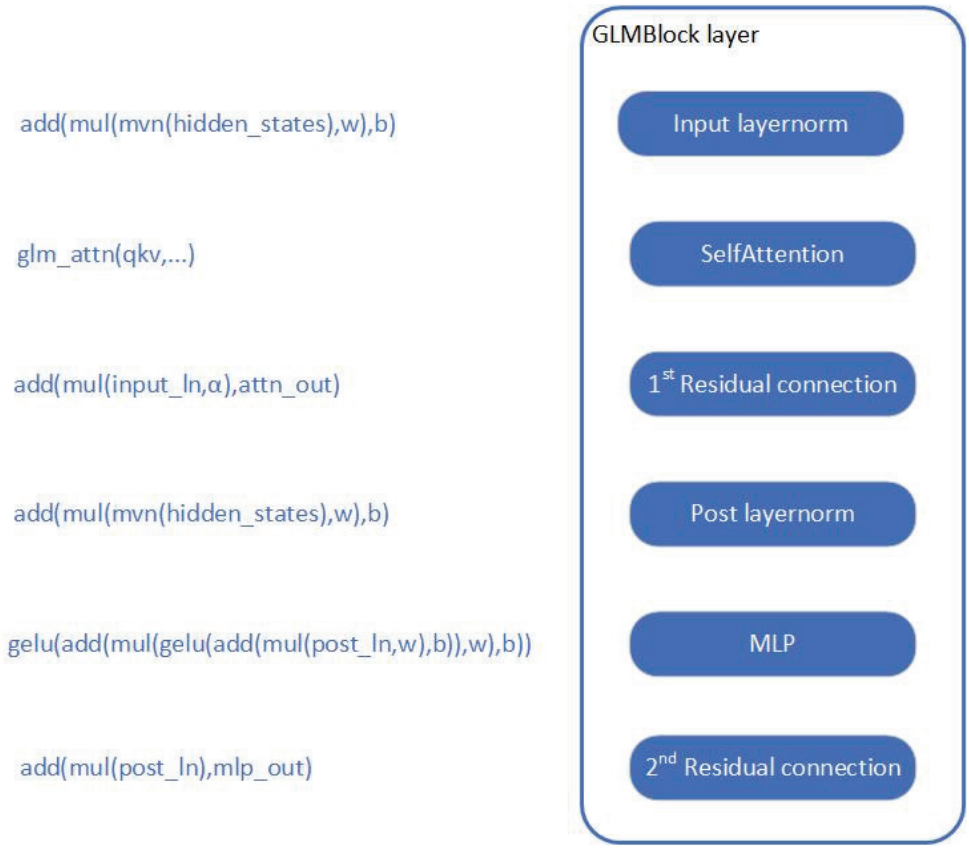

图 2. 为 ChatGLM 构建 OpenVINO™ stateful 模型

关于如何构建 OpenVINO™ stateful 模型，可参阅以下文档：
https://docs.openvino.ai/2022.3/openvino_docs_OV_UG_network_state_intro.html
OpenVINO™ 还提供了模型创建样本，以展示如何通过 opset 构建模型。
https://github.com/openvinotoolkit/openvino/blob/master/samples/cpp/model_creation_sample/main.cpp

ChatGLM 的自定义注意力机制是本文所关注和优化的部分。主要思路是：构建全局上下文结构体，用于在模型内部追加并保存每一轮迭代后的 pastKV 的结果，这样减少了 pastKV 作为模型输入输出的拷贝开销，同时使用内联优化以实现 Rotary Embedding 和多头注意力机制 (Multi-Head Attentions)。

英特尔® AMX 是内置在第四代英特尔® 至强® 可扩展处理器中的矩阵乘法加速器，能够更快速地处理 bf16 或 int8 数据类型的矩阵乘加运算，通过加速张量处理，显著提高推理和训练性能。借助英特尔® AMX 内联指令（用于加速计算的单指令多操作），实现了对 ChatGLM 模型中 Attention，Rotary Embedding 等算子的高度优化，并且使用 bf16 指令进行乘加操作，在保证浮点指数位精度的同时提高运算效率。

与此同时，本方案还使用 int8 精度来压缩全连接层的权重，在实时计算中将使用 bf16 进行计算。因此，无需通过训练后量化 (PTQ) 或量化感知训练（QAT）对模型进行低精度处理。模型压缩方法可以降低模型存储空间，减少内存带宽的负载，因为计算仍然使用浮点，不会造成溢出，不会对模型精度造成损失。

为 ChatGLM 创建 OpenVINO™ stateful 模型

请依照下方示例配置软硬件环境，并按照以下步骤优化 ChatGLM:

硬件要求

第四代英特尔® 至强® 可扩展处理器（代号 Sapphire Rapids）或其后续的、仍内置英特尔® AMX 的产品

软件验证环境

Ubuntu 22.04.1 LTS

面向 OpenVINO™ Runtime Python API 的 Python 3.10.11

用于构建 OpenVINO™ Runtime 的 GCC 11.3.0

cmake 3.26.4

构建 OpenVINO™ 源码

- 安装系统依赖并设置环境
- 创建并启用 Python 虚拟环境

```
$ conda create -n ov_py310 python=3.10 -y
$ conda activate ov_py310
```

- 安装 Python 依赖

```
$ pip install protobuf transformers==4.30.2 cpm_kernels torch>=2.0 sentencepiece pandas
```

- 使用 GCC 11.3.0 编译 OpenVINO™
- 克隆 OpenVINO™ 并升级子模块

```
$ git clone https://github.com/luo-cheng2021/openvino.git -b luocheng/chatglm_custom
$ cd openvino && git submodule update --init --recursive
```

- 安装 Python 环境依赖，以构建 Python Wheel

```
$ python -m pip install -U pip
```

```
$ python -m pip install -r ./src/bindings/python/src/compatibility/openvino/requirements-dev.txt
$ python -m pip install -r ./src/bindings/python/wheel/requirements-dev.txt
```

- 创建编译目录

```
$ mkdir build && cd build
```

- 使用 CMake 编译 OpenVINO™

```
$ cmake .. -DENABLE_LLMDNN=ON \
  -DBUILD_PYTHON_TESTS=ON \
  -DENABLE_CPU_DEBUG_CAPS=OFF \
  -DENABLE_DEBUG_CAPS=OFF \
  -DCMAKE_BUILD_TYPE=Release \
  -DENABLE_INTEL_MYRIAD_COMMON=OFF \
  -DENABLE_INTEL_GNA=OFF \
  -DENABLE_OPENCV=OFF \
  -DENABLE_CPPLINT=ON \
  -DENABLE_CPPLINT_REPORT=OFF \
  -DENABLE_NCC_STYLE=OFF \
  -DENABLE_TESTS=ON \
  -DENABLE_OV_CORE_UNIT_TESTS=OFF \
  -DENABLE_INTEL_CPU=ON \
  -DENABLE_INTEL_GPU=OFF \
  -DENABLE_AUTO=OFF \
  -DENABLE_AUTO_BATCH=OFF \
  -DENABLE_MULTI=OFF \
  -DENABLE_HETERO=OFF \
  -DENABLE_INTEL_GNA=OFF \
  -DENABLE_PROFILING_ITT=ON\
  -DENABLE_SAMPLES=ON \
  -DENABLE_PYTHON=ON \
  -DENABLE_TEMPLATE=OFF \
  -DENABLE_OV_ONNX_FRONTEND=OFF \
  -DENABLE_OV_PADDLE_FRONTEND=OFF \
  -DENABLE_OV_PYTORCH_FRONTEND=OFF \
  -DENABLE_OV_TF_FRONTEND=OFF \
  -DENABLE_OPENVINO_DEBUG=OFF \
  -DENABLE_CPU_DEBUG_CAPS=ON \
  -DCMAKE_INSTALL_PREFIX=`pwd`/install \
```

```
  -
DCMAKE_INSTALL_RPATH=`pwd`/install/runtime/3rdparty/tbb/lib:`pwd`/install/runtime/3rdparty/hddl/lib:`pwd`/install/runtime/lib/intel64 \
  -Dgflags_Dir=`pwd`/../thirdparty/gflags/gflags/cmake
$ make --jobs=$(nproc --all)
$ make install
```

- 安装针对 OpenVINO™ Runtime 和 openvino-dev 工具构建好的 Python Wheel

```
$ pip install ./install/tools/openvino*.whl
```

- 检查系统 GCC 版本和 Conda Runtime GCC 版本。如下所示，如果系统 GCC 版本高于 Conda GCC 版本，请升级 Conda GCC 至相同版本，以满足 OpenVINO™ Runtime 的需求。（可选）

```
##check system (OpenVINO compiling env) gcc version
$ gcc --version
gcc (Ubuntu 11.3.0-1ubuntu1~22.04.1) 11.3.0
```

```
##check conda python (runtime env for OpenVINO later) gcc version
$ python
Python 3.10.11 (main, May 16 2023, 00:28:57) [GCC 11.2.0] on linux
```

```
##If sys gcc ver > conda gcc ver, upgrade conda gcc ver -> sys gcc ver
$ conda install -c conda-forge gcc=11.3.0
```

- 将 PyTorch 模型转为 OpenVINO™ IR

```
$ cd ..
$ python tools/gpt/gen_chatglm.py /path/to/pytorch/model /path/to/ov/IR
```

使用 OpenVINO™ Runtime API 为 ChatGLM 构建推理流水线

本文提供了使用 Transformer 和 OpenVINO™ Runtime API 构建推理流水线的样本。首先，在 *test_chatglm.py* 中，创建一个由 *transformers.PreTrainedModel* 衍生的新类。然后，通过使用 OpenVINO™ Runtime Python API 构建模型推理流水线来更新转发函数。其他成员函数则迁移自 modeling_chatglm.py 的 *ChatGLMForConditionalGeneration*，如此一来，即可确保输入准备工作、*set_random_seed*、分词器/连接器 (tokenizer/detokenizer) 以及余下的流水线操作能够与原始模型的源码保持一致。
如需启用 int8 权重压缩，只需设置简单的环境变量 USE_INT8_WEIGHT=1。这是因为在模型生成阶段，已使用 int8 对全连接层的权重进行了压缩，因此模型可在之后的运行过程中直接使用 int8 权重进行推理，从而免除了通过框架或量化工具压缩模型的步骤。

请按照以下步骤使用 OpenVINO™ Runtime 流水线测试 ChatGLM：

- 运行 bf16 模型

```
$ python3 tools/gpt/test_chatglm.py /path/to/pytorch/model /path/to/ov/IR --use=ov
```

- 运行 int8 模型

```
$ USE_INT8_WEIGHT=1 python test_chatglm.py /path/to/pytorch/model /path/to/ov/IR --use=ov
```

权重压缩：降低内存带宽使用率，提升推理速度

本文采用了 Vtune 对模型权重数值精度分别为 bf16 和 int8 的内存带宽使用率（图 3 和图 4）以及 CPI 率进行了性能对比分析（表 1）。结果发现：当模型权重数值精度压缩至 int8 时，可同时降低内存带宽使用率和 CPI 率。

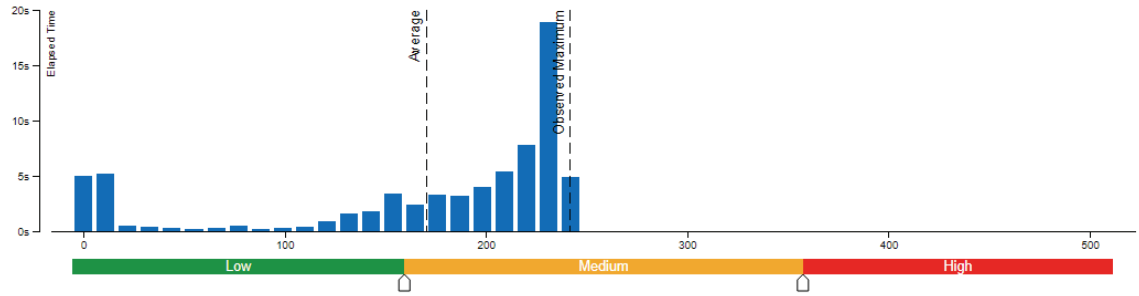

图 3. 模型权重数值精度为 bf16 时的内存带宽使用率

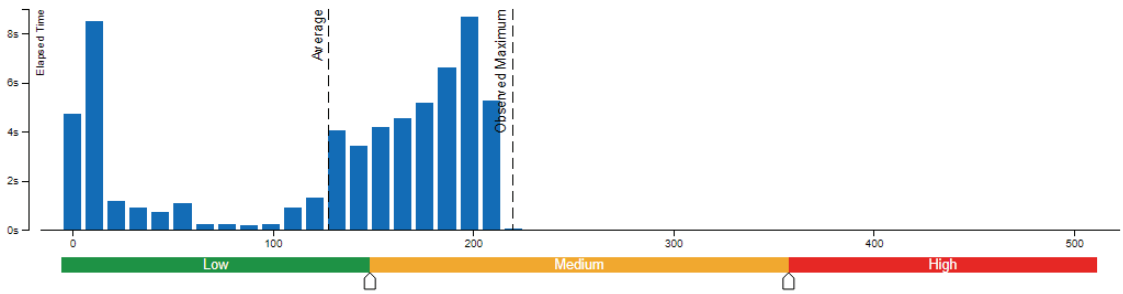

图 4. 模型权重数值精度为 int8 时的内存带宽使用率

模型权重数值精度	bf16	int8
CPI 率	10.766	1.175

表 1.采用不同模型权重数值精度时的 CPI 率

每条指令消耗的时钟周期 (Clockticks per Instruction Retired, CPI) 事件率，也称为“平均指令周期数 (Cycles per Instruction)”，是基于硬件事件抽样收集的基础性能指标之一，在抽样模式下也称为“性能监控计数器 (PMC) 分析”。该比率计算方式为：用处于非停机状态的处理器时钟周期数 (Clockticks) 除以已消耗指令数。每个处理器用于计算时钟周期数和已消耗指令数的确切事件可能并不相同，但 VTune Profiler 可辨别和使用正确的数量。

CPI <1时，通常为采用指令密集型代码的应用，而 CPI >1则可能是停滞时钟周期密集型应用，也可能是内存密集型应用。由此，我们可以得出结论，类似 chatGLM 等语言模型对内存带宽的要求非常高，性能往往受到内存操作或带宽的限制。很多场景下，消除内存操作的负载，性能会因此获得大幅收益。在优化此类模型时，如何在不影响精度的同时对模型进行压缩或轻量化处理是一项不可或缺的技巧。除此之外，在异构平台和框架上进行部署，还涉及到减少内存/设备存储之间的数据搬运等优化思路。因此，在压缩模型的同时，还需要考虑对原始 pytorch 模型推理 forward/generates 等函数流水线的优化，而 OpenVINO™ 在优化模型自身的同时，还将流水线的优化思路体现在修改模型结构中（将 KV cache 保存在模型内部），通过优化 Optimum-intel 等框架的流水线，减少内存拷贝和数据搬运。

结论

笔者根据上述方法重新设计执行图并优化了 GLMBlock，消除了 ChatGLM 模型输入和输出之间的内存副本，且模型运行高效。随着 OpenVINO™ 的不断升级，本方案的优化工作也将得到推广并集成至正式发布的版本中。这将有助于扩展更多的大语言模型用例。敬请参考 OpenVINO™ 官方版本[iii] 和 Optimum-intel OpenVINO™ 后端[iv]，获取有关大语言模型的官方高效支持。

作者简介：

英特尔® OpenVINO™ 开发工具客户支持工程师赵桢和邹文艺，英特尔® OpenVINO™ 开发工具 AI 框架工程师罗成和李亭謇，都在从事 AI 软件工具开发与优化工作。

[i] https://www.intel.cn/content/www/cn/zh/events/accelerate-with-xeon.html
[ii] https://huggingface.co/THUDM/chatglm-6b/blob/main/config.json
[iii] https://www.intel.cn/content/www/cn/zh/developer/tools/openvino-toolkit/overview.html
[iv] https://huggingface.co/docs/optimum/main/en/intel/index