Mobile apps are very sensitive to their download sizes as every increase in KB may result in a user number decrease. Flutter engine (`libflutter.so`) has to be included in every Flutter app and it has a size of several MBs. So we'll try to reduce its size by using MLGO. It's different from the previous Flutter attempt of reducing sizes as MLGO does not require any code or dependency removals.

Reducing engine size with MLGO needs to 1) train a model once, 2) apply that model to compile the Flutter engine. Note that model training does not need to happen too frequently - the model should 'hold up' to code changes over weeks/months. On Ubuntu, do the following:

- **Follow the Setting-up-the-Engine-development-environment**. To check if this step is successful, try compiling for Android. For size comparisons, we recommend using the release Android build `./flutter/tools/gn --android --runtime-mode=release --no-goma` . (Option `--no-goma` is needed if you're not a Googler, or if you're using a custom Clang as we'll do later with MLGO.)

- **Set up MLGO LLVM**. (The steps are adapted from MLGO demo.)

  1. Prerequisites: `sudo apt-get install cmake ninja-build lld` .
  2. Create a root directory for everything MLGO related `mkdir ~/mlgo && export MLGO_DIR=~/mlgo`
  3. Clone MLGO repo `cd $MLGO_DIR && git clone https://github.com/google/ml-compiler-opt.git`
  4. Tensorflow dependencies

     ```
     cd $MLGO_DIR
     sudo apt-get install python3-pip
     python3 -m pip install --upgrade pip
     python3 -m pip install --user -r ml-compiler-opt/requirements.txt

     TF_PIP=$(python3 -m pip show tensorflow | grep Location | cut -d ' ' -f 2)

     export TENSORFLOW_AOT_PATH="${TF_PIP}/tensorflow"

     mkdir $MLGO_DIR/tensorflow
     export TENSORFLOW_C_LIB_PATH=$MLGO_DIR/tensorflow

     wget --quiet https://storage.googleapis.com/tensorflow/libtensorflow/libtensorflow-cpu-linux-x86_64-1.15.0.tar.gz

     tar xfz libtensorflow-cpu-linux-x86_64-1.15.0.tar.gz -C "${TENSORFLOW_C_LIB_PATH}"
     ```

  5. Clone llvm-project

     ```
     cd $MLGO_DIR && git clone https://github.com/llvm/llvm-project.git
     export LLVM_SRCDIR=$MLGO_DIR/llvm-project
     export LLVM_INSTALLDIR=$MLGO_DIR/llvm-install
     ```

  6. Build LLVM

```
cd ${LLVM_SRCDIR}
mkdir build
cd build
cmake -G Ninja \
  -DLLVM_ENABLE_LTO=OFF \
  -DCMAKE_INSTALL_PREFIX= \
  -DTENSORFLOW_C_LIB_PATH=${TENSORFLOW_C_LIB_PATH} \
  -DCMAKE_INSTALL_RPATH_USE_LINK_PATH=On \
  -C ${LLVM_SRCDIR}/clang/cmake/caches/Fuchsia-stage2.cmake \
  ${LLVM_SRCDIR}/llvm

ninja distribution
DESTDIR=${LLVM_INSTALLDIR} ninja install-distribution-stripped
```

- **Build Flutter engine for MLGO training**

```
# Set your engine dir appropriately if it's not in the default location
export ENGINE_DIR=~/flutter/engine/src
cd $ENGINE_DIR

sed -i \
  's/cflags += lto_flags/cflags += lto_flags + ["-Xclang", "-fembed-bitcode=all"]/' \
  build/config/compiler/BUILD.gn

sed -i \
  "s/prefix = rebase_path(\"\/\/buildtools\/\$host_dir\/clang\/bin\", root_build_dir)/prefix = \"${LLVM_INSTALLDIR//\//\\/}\/bin\"/" \
  build/toolchain/android/BUILD.gn

./flutter/tools/gn --android --runtime-mode=release --no-goma --no-lto
ninja -C out/android_release
```

- **Train the model**

```
export CORPUS=$MLGO_DIR/corpus
cd $MLGO_DIR/ml-compiler-opt
python3 compiler_opt/tools/extract_ir.py \
  --cmd_filter="^-Oz$" \
  --input=$ENGINE_DIR/out/compile_commands.json \
  --input_type=json \
  --llvm_objcopy_path=$LLVM_INSTALLDIR/bin/llvm-objcopy \
  --output_dir=$CORPUS

export DEFAULT_TRACE=$MLGO_DIR/default_trace
export WARMSTART_OUTPUT_DIR=$MLGO_DIR/warmstart
export OUTPUT_DIR=$MLGO_DIR/model

rm -rf $DEFAULT_TRACE && \
  PYTHONPATH=$PYTHONPATH:. python3 \
   compiler_opt/tools/generate_default_trace.py \
   --data_path=$CORPUS \
   --output_path=$DEFAULT_TRACE \
   --compile_task=inlining \
   --clang_path=$LLVM_INSTALLDIR/bin/clang \
   --llvm_size_path=$LLVM_INSTALLDIR/bin/llvm-size \
   --sampling_rate=0.2

rm -rf $WARMSTART_OUTPUT_DIR && \
  PYTHONPATH=$PYTHONPATH:. python3 \
  compiler_opt/rl/train_bc.py \
  --root_dir=$WARMSTART_OUTPUT_DIR \
  --data_path=$DEFAULT_TRACE \
  --
gin_files=compiler_opt/rl/inlining/gin_configs/behavioral_cloning_nn_agent.gin


# The following will take about half a day.
rm -rf $OUTPUT_DIR && \
  PYTHONPATH=$PYTHONPATH:. python3 \
  compiler_opt/rl/train_locally.py \
  --root_dir=$OUTPUT_DIR \
  --data_path=$CORPUS \
  --clang_path=$LLVM_INSTALLDIR/bin/clang \
  --llvm_size_path=$LLVM_INSTALLDIR/bin/llvm-size \
  --num_modules=100 \
  --gin_files=compiler_opt/rl/inlining/gin_configs/ppo_nn_agent.gin \
  --
gin_bindings=train_eval.warmstart_policy_dir=\"$WARMSTART_OUTPUT_DIR/saved_polic
```

- **Build LLVM with the trained model**

```
cd $LLVM_SRCDIR
rm -rf llvm/lib/Analysis/models/inliner/*
cp -rf $OUTPUT_DIR/saved_policy/* llvm/lib/Analysis/models/inliner/

mkdir build-release
cd build-release
cmake -G Ninja \
  -DLLVM_ENABLE_LTO=OFF \
  -DCMAKE_INSTALL_PREFIX= \
  -DTENSORFLOW_AOT_PATH=${TENSORFLOW_AOT_PATH} \
  -C ${LLVM_SRCDIR}/clang/cmake/caches/Fuchsia-stage2.cmake \
  ${LLVM_SRCDIR}/llvm

export LLVM_INSTALLDIR_RELEASE=$LLVM_INSTALLDIR-release
ninja distribution
DESTDIR=${LLVM_INSTALLDIR_RELEASE} ninja install-distribution-stripped
```

- **Build Flutter engine using LLVM with the trained model**

```
cd $ENGINE_DIR
git stash # Undo previous changes for model training

sed -i \
  's/cflags += lto_flags/cflags += lto_flags + ["-mllvm", "-enable-ml-inliner=release"]/' \
  build/config/compiler/BUILD.gn

sed -i \
  "s/prefix = rebase_path(\"\/\/buildtools\/\$host_dir\/clang\/bin\", root_build_dir)/prefix = \"${LLVM_INSTALLDIR_RELEASE//\//\\/}\/bin\"/" \
  build/toolchain/android/BUILD.gn

./flutter/tools/gn --android --runtime-mode=release --no-goma --no-lto
ninja -C out/android_release libflutter.so
```

- **Compare**. To compare the engine size with or without MLGO, one can add or remove the `["-mllvm", "-enable-ml-inliner=release"]` flags in `build/config/compiler/BUILD.gn`, compile the engine, and check the size of `out/android_release/lib.stripped/libflutter.so`. As end-users will download zipped engine, we also recommend comparing its zipped size.

```
export ENGINE_LIB_DIR=$ENGINE_DIR/out/android_release/lib.stripped

cd $ENGINE_DIR
./flutter/tools/gn --android --runtime-mode=release --no-goma --no-lto
ninja -C out/android_release libflutter.so
cd $ENGINE_LIB_DIR
mv libflutter.so libflutter.ml_nolto.so
zip libflutter.ml_nolto.so.zip libflutter.ml_nolto.so

cd $ENGINE_DIR
./flutter/tools/gn --android --runtime-mode=release --no-goma
ninja -C out/android_release libflutter.so
cd $ENGINE_LIB_DIR
mv libflutter.so libflutter.ml_lto.so
zip libflutter.ml_lto.so.zip libflutter.ml_lto.so

# Remove the ML flags to disable ML.
cd $ENGINE_DIR
sed -i \
  's/cflags += lto_flags + \["-mllvm", "-enable-ml-inliner=release"\]/cflags += lto_flags/' \
  build/config/compiler/BUILD.gn

cd $ENGINE_DIR
./flutter/tools/gn --android --runtime-mode=release --no-goma --no-lto
ninja -C out/android_release libflutter.so
cd $ENGINE_LIB_DIR
mv libflutter.so libflutter.noml_nolto.so
```

```
zip libflutter.noml_nolto.so.zip libflutter.noml_nolto.so

cd $ENGINE_DIR
./flutter/tools/gn --android --runtime-mode=release --no-goma
ninja -C out/android_release libflutter.so
cd $ENGINE_LIB_DIR
mv libflutter.so libflutter.noml_lto.so
zip libflutter.noml_lto.so.zip libflutter.noml_lto.so

ls -l
```

Here's the table of size comparisons for engine version b9ecd8a.

Flutter engine size comparison	ML_LTO	ML_NOLTO	NOML_LTO	NOML_NOLTO
unzipped size (bytes)	6270960	6338580	6312012	6577684
zipped size (bytes)	3586091	**3577604**	3606484	3689468
unzipped size change over NOML_LTO	-0.65%	0.4%	0	4.21%
zipped size change over NOML_LTO	-0.57%	**-0.80%**	0	2.3%
unzipped size change over NOML_NOLTO	-4.66%	-3.64%	-4.04%	0
zipped size change over NOML_NOLTO	-2.80%	-3.03%	-2.25%	0

Conclusion

As shown in the table above, for the zipped size, the winner here is the ML_NOLTO version which is even smaller than the ML_LTO version. It has a 0.8% reduction over our previous art of NOML_LTO.

The ML_LTO version is not very good because currently the model can only be trained without LTO. MLGO is planning to allow ThinLTO in their training. Hopefully, it will help achieve the MLGO's normal reduction of 3%-5% (e.g., ML_NOLTO vs NOML_NOLTO) when the training and final build are in the same condition.