# TensorFlow™ with LIBXSMM

This document is an early recipe for building and running TensorFlow with LIBXSMM. The amount of covered code paths as well as the performance of these code paths will be improved as the integration progresses. This means in particular that executing TensorFlow for any real workload (or benchmark) beside of the cases shown below **may not use LIBXSMM at all** (same is/remains true for any system without Intel AVX2 instruction set extension). Relying on the master revision of TensorFlow is perfectly fine since the integration work is merged on a fairly frequent basis. However, to capture the latest revision of the integration one may rely on:

```
git clone https://github.com/benoitsteiner/tensorflow-xsmm.git
wget https://github.com/hfp/libxsmm/archive/master.zip
sha256sum libxsmm-master.zip
```

In order to try LIBXSMM's master revision, the file `tensorflow/workspace.bzl` can be adjusted using the SHA256 value from above:

```
  native.new_http_archive(
      name = "libxsmm_archive",
      urls = [
          "https://github.com/hfp/libxsmm/archive/master.zip",
      ],
      sha256 = "<sha256sum libxsmm-master.zip>",
      strip_prefix = "libxsmm-master",
      build_file = str(Label("//third_party:libxsmm.BUILD")),
  )
```

Beside of regular prerequisites, nothing else is needed in order to use TensorFlow with LIBXSMM. However, at least in some cases (NFS-hosted directory, etc.) it may help to adjust the `configure` script (change `bazel clean --expunge` into `bazel clean --expunge_async`).

```
cd tensorflow-xsmm
./configure
```

There are two aspects of LIBXSMM enabled within TensorFlow: (1) sparse CNN, and (2) CNN. To build and test the sparse routines:

```
bazel build -c opt --copt=-mavx2 --copt=-mfma --verbose_failures \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //tensorflow/core/kernels:sparse_matmul_op_test

bazel run -c opt --copt=-mavx2 --copt=-mfma \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //tensorflow/python/kernel_tests:sparse_matmul_op_test

bazel-bin/tensorflow/core/kernels/sparse_matmul_op_test --benchmarks=all
bazel-bin/tensorflow/core/kernels/sparse_matmul_op_test
```

To build and test the regular CNN routines:

```
bazel build -c opt --copt=-mavx2 --copt=-mfma --verbose_failures \
  --define tensorflow_xsmm=1 --define eigen_xsmm=1 --define tensorflow_xsmm_backward=1 \
  //tensorflow/core/kernels:conv_ops_test

bazel-bin/tensorflow/core/kernels/conv_ops_test
```

Please note that `--define tensorflow_xsmm=1`, `--define eigen_xsmm=1`, and `--define tensorflow_xsmm_backward=1` are not actually needed for each of the above cases, but are supplied for consistency. More important, `--copt=-mavx2 --copt=-mfma` are only suitable for Intel AVX2 capable systems. LIBXSMM does not impose to build for a specific code path, which is always true for JIT-enabled code paths. However, when using the Intel Compiler or GCC 4.9 (or later), even some non-JIT code is CPUID-dispatched. In turn, older GCC compilers may be hit by some performance penalty when building without the necessary target flags. In any case it makes sense to simply build at least with `--copt=-mavx`. As a further note, the LIBXSMM supports Intel AVX2 as the baseline code path for all JIT-generated DNN-code (SMM domain also supports AVX). For Intel AVX-512 (on top of AVX2), the foundational instructions are sufficient in many cases, but for the sparse domain the Core-flavor is a prerequisite ("Skylake server" or SKX), and VNNI/QFMA instructions are honored on Intel Xeon Phi code-named "Knights Mill" (KNM).