Buddy-CAAS: Compiler As A Service for MLIR

Speaker: Hongbin Zhang | hongbin2019@iscas.ac.cn

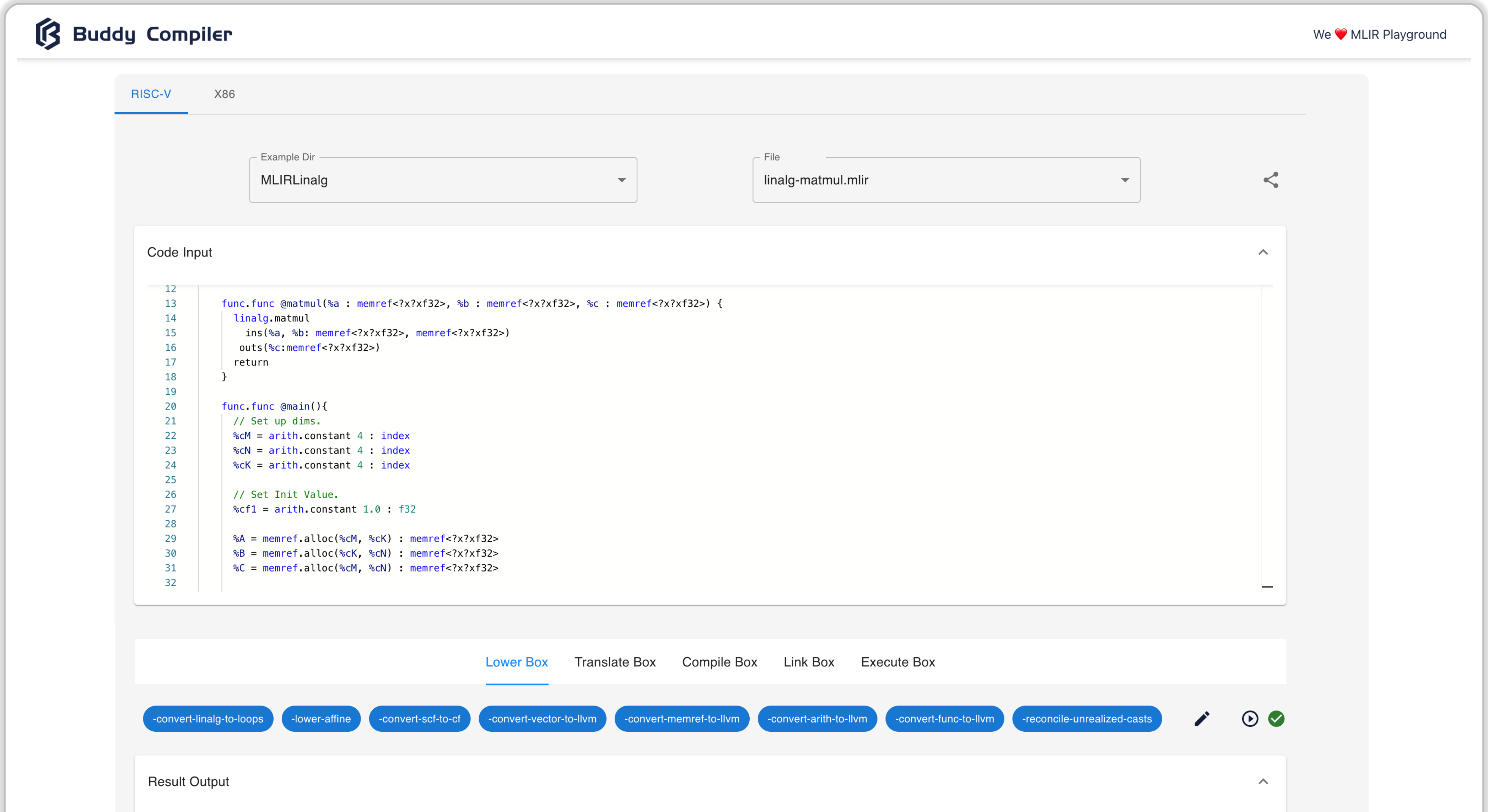
Authors: Zikang Liu (ICT), Hongbin Zhang (ISCAS), Bofan Zhang (USYD)

Buddy Compiler

Compiler As A Service for MLIR (https://buddy.isrc.ac.cn/)

Buddy Compiler

Buddy-CAAS serves MLIR users and developers to configure the pass pipeline and demonstrate on multiple backends quickly and smoothly.



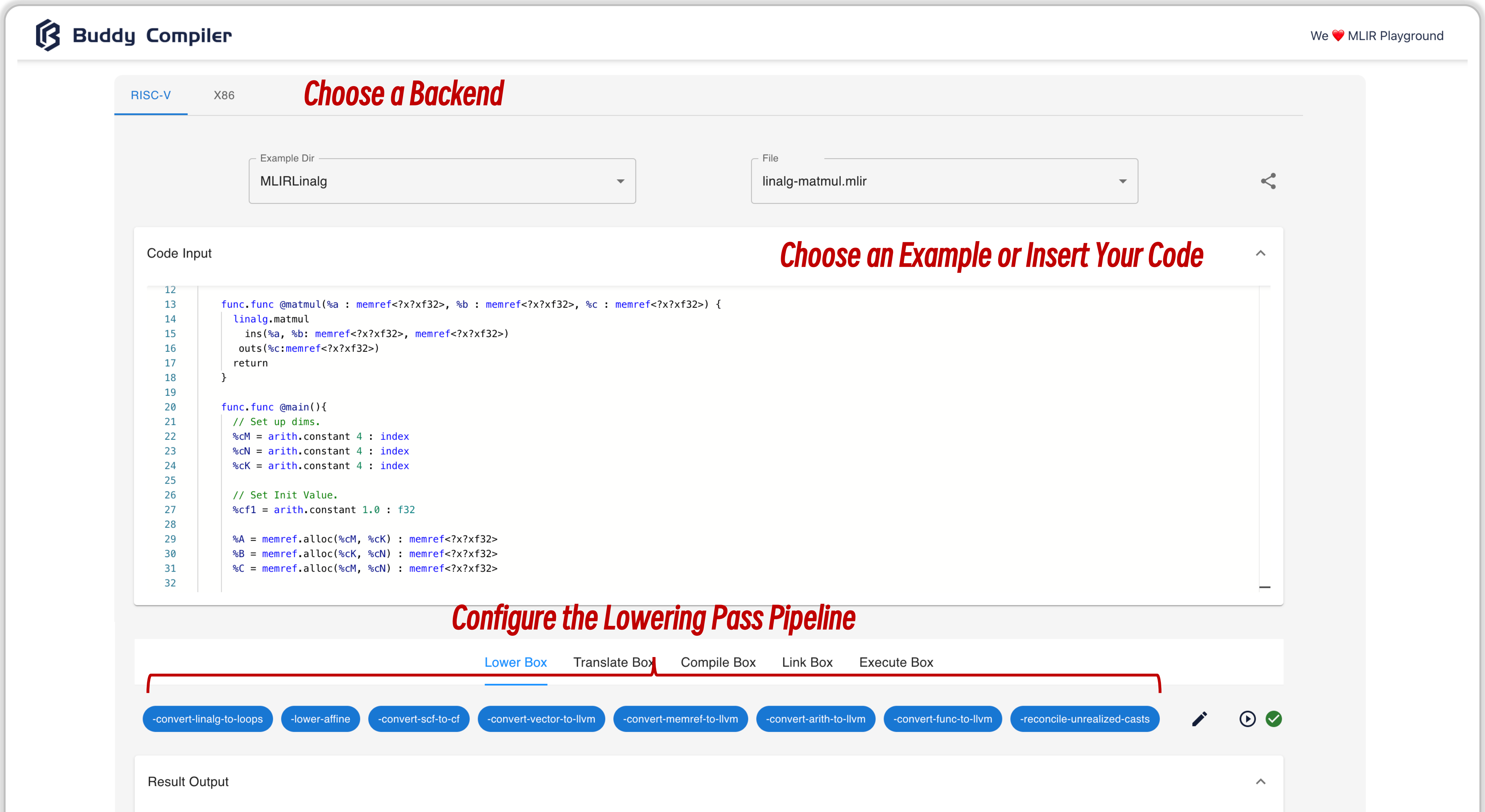
```
func.func @matmul(%a : memref<?x?xf32>, %b : memref<?x?xf32>, %c : memref<?x?xf32>) {
  linalg.matmul
    ins(%a, %b: memref<?x?xf32>, memref<?x?xf32>)
   outs(%c:memref<?x?xf32>)
  return
}

func.func @main(){
  // Set up dims.
  %cM = arith.constant 4 : index
  %cN = arith.constant 4 : index
  %cK = arith.constant 4 : index

  // Set Init Value.
  %cf1 = arith.constant 1.0 : f32

  %A = memref.alloc(%cM, %cK) : memref<?x?xf32>
  %B = memref.alloc(%cK, %cN) : memref<?x?xf32>
  %C = memref.alloc(%cM, %cN) : memref<?x?xf32>
```

-convert-linalg-to-loops -lower-affine -convert-scf-to-cf -convert-vector-to-llvm -convert-memref-to-llvm -convert-arith-to-llvm -convert-func-to-llvm -reconcile-unrealized-casts

Compiler As A Service for MLIR (https://buddy.isrc.ac.cn/)

Buddy Compiler

Buddy-CAAS serves MLIR users and developers to configure the pass pipeline and demonstrate on multiple backends quickly and smoothly.


Buddy Compiler

We ❤️ MLIR Playground

RISC-V X86

Choose a Backend

Example Dir: MLIRLinalg

File: linalg-matmul.mlir

Code Input

Choose an Example or Insert Your Code

```
func.func @matmul(%a : memref<?x?xf32>, %b : memref<?x?xf32>, %c : memref<?x?xf32>) {
  linalg.matmul
    ins(%a, %b: memref<?x?xf32>, memref<?x?xf32>)
   outs(%c:memref<?x?xf32>)
  return
}

func.func @main(){
  // Set up dims.
  %cM = arith.constant 4 : index
  %cN = arith.constant 4 : index
  %cK = arith.constant 4 : index

  // Set Init Value.
  %cf1 = arith.constant 1.0 : f32

  %A = memref.alloc(%cM, %cK) : memref<?x?xf32>
  %B = memref.alloc(%cK, %cN) : memref<?x?xf32>
  %C = memref.alloc(%cM, %cN) : memref<?x?xf32>

```

Configure the Lowering Pass Pipeline

Lower Box | Translate Box | Compile Box | Link Box | Execute Box

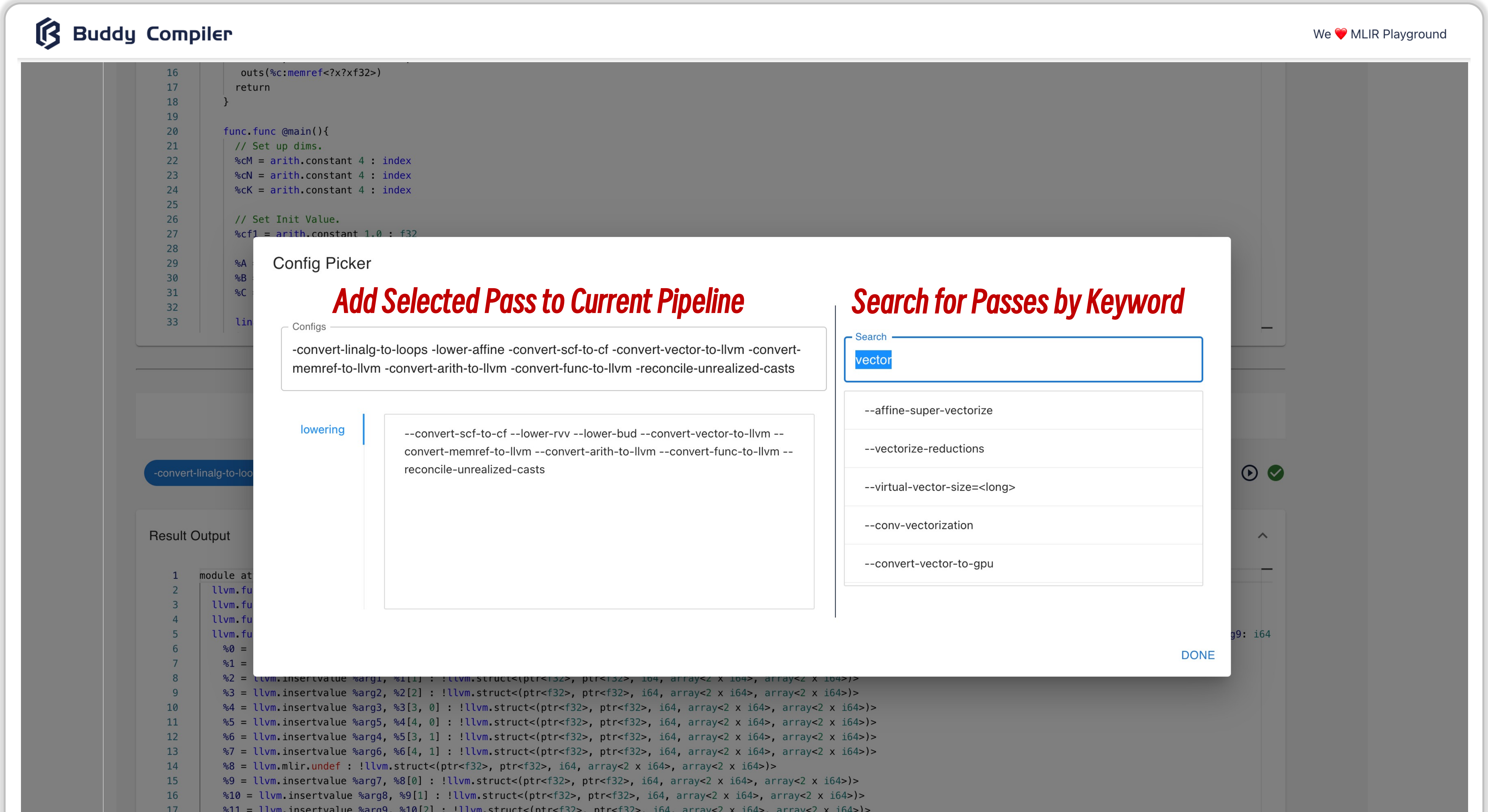
-convert-linalg-to-loops -lower-affine -convert-scf-to-cf -convert-vector-to-llvm -convert-memref-to-llvm -convert-arith-to-llvm -convert-func-to-llvm -reconcile-unrealized-casts

Result Output

Compiler As A Service for MLIR (https://buddy.isrc.ac.cn/)

Buddy Compiler

Buddy-CAAS serves MLIR users and developers to configure the pass pipeline and demonstrate on multiple backends quickly and smoothly.


Buddy Compiler

We ❤️ MLIR Playground

```
      outs(%c:memref<?x?xf32>)
     return
   }

   func.func @main(){
     // Set up dims.
     %cM = arith.constant 4 : index
     %cN = arith.constant 4 : index
     %cK = arith.constant 4 : index

     // Set Init Value.
     %cf1 = arith.constant 1.0 : f32

     %A
     %B
     %C

     lin
```

Config Picker

Add Selected Pass to Current Pipeline

Configs

-convert-linalg-to-loops -lower-affine -convert-scf-to-cf -convert-vector-to-llvm -convert-memref-to-llvm -convert-arith-to-llvm -convert-func-to-llvm -reconcile-unrealized-casts

lowering

--convert-scf-to-cf --lower-rvv --lower-bud --convert-vector-to-llvm --convert-memref-to-llvm --convert-arith-to-llvm --convert-func-to-llvm --reconcile-unrealized-casts

Search for Passes by Keyword

Search

vector

--affine-super-vectorize

--vectorize-reductions

--virtual-vector-size=<long>

--conv-vectorization

--convert-vector-to-gpu

DONE

-convert-linalg-to-loo

Result Output

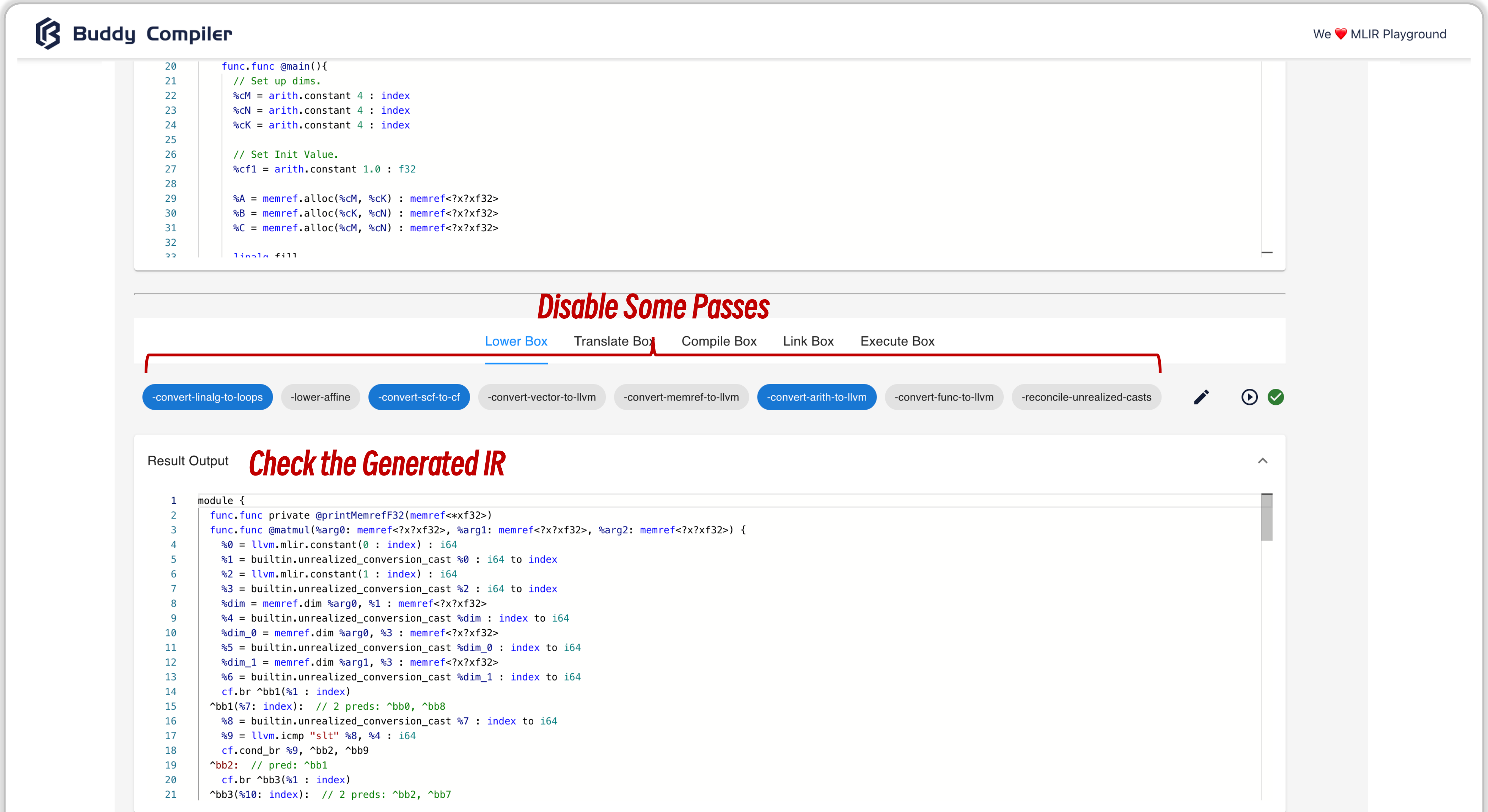
```
module at
  llvm.fu
  llvm.fu
  llvm.fu
  llvm.fu
    %0 =
    %1 =
    %2 = llvm.insertvalue %arg1, %1[1] : !llvm.struct<(ptr<f32>, ptr<f32>, i64, array<2 x i64>, array<2 x i64>)>
    %3 = llvm.insertvalue %arg2, %2[2] : !llvm.struct<(ptr<f32>, ptr<f32>, i64, array<2 x i64>, array<2 x i64>)>
    %4 = llvm.insertvalue %arg3, %3[3, 0] : !llvm.struct<(ptr<f32>, ptr<f32>, i64, array<2 x i64>, array<2 x i64>)>
    %5 = llvm.insertvalue %arg5, %4[4, 0] : !llvm.struct<(ptr<f32>, ptr<f32>, i64, array<2 x i64>, array<2 x i64>)>
    %6 = llvm.insertvalue %arg4, %5[3, 1] : !llvm.struct<(ptr<f32>, ptr<f32>, i64, array<2 x i64>, array<2 x i64>)>
    %7 = llvm.insertvalue %arg6, %6[4, 1] : !llvm.struct<(ptr<f32>, ptr<f32>, i64, array<2 x i64>, array<2 x i64>)>
    %8 = llvm.mlir.undef : !llvm.struct<(ptr<f32>, ptr<f32>, i64, array<2 x i64>, array<2 x i64>)>
    %9 = llvm.insertvalue %arg7, %8[0] : !llvm.struct<(ptr<f32>, ptr<f32>, i64, array<2 x i64>, array<2 x i64>)>
    %10 = llvm.insertvalue %arg8, %9[1] : !llvm.struct<(ptr<f32>, ptr<f32>, i64, array<2 x i64>, array<2 x i64>)>
    %11 = llvm.insertvalue %arg9, %10[2] : !llvm.struct<(ptr<f32>, ptr<f32>, i64, array<2 x i64>, array<2 x i64>)>
```

g9: i64

Compiler As A Service for MLIR (https://buddy.isrc.ac.cn/)

Buddy Compiler

Buddy-CAAS serves MLIR users and developers to configure the pass pipeline and demonstrate on multiple backends quickly and smoothly.



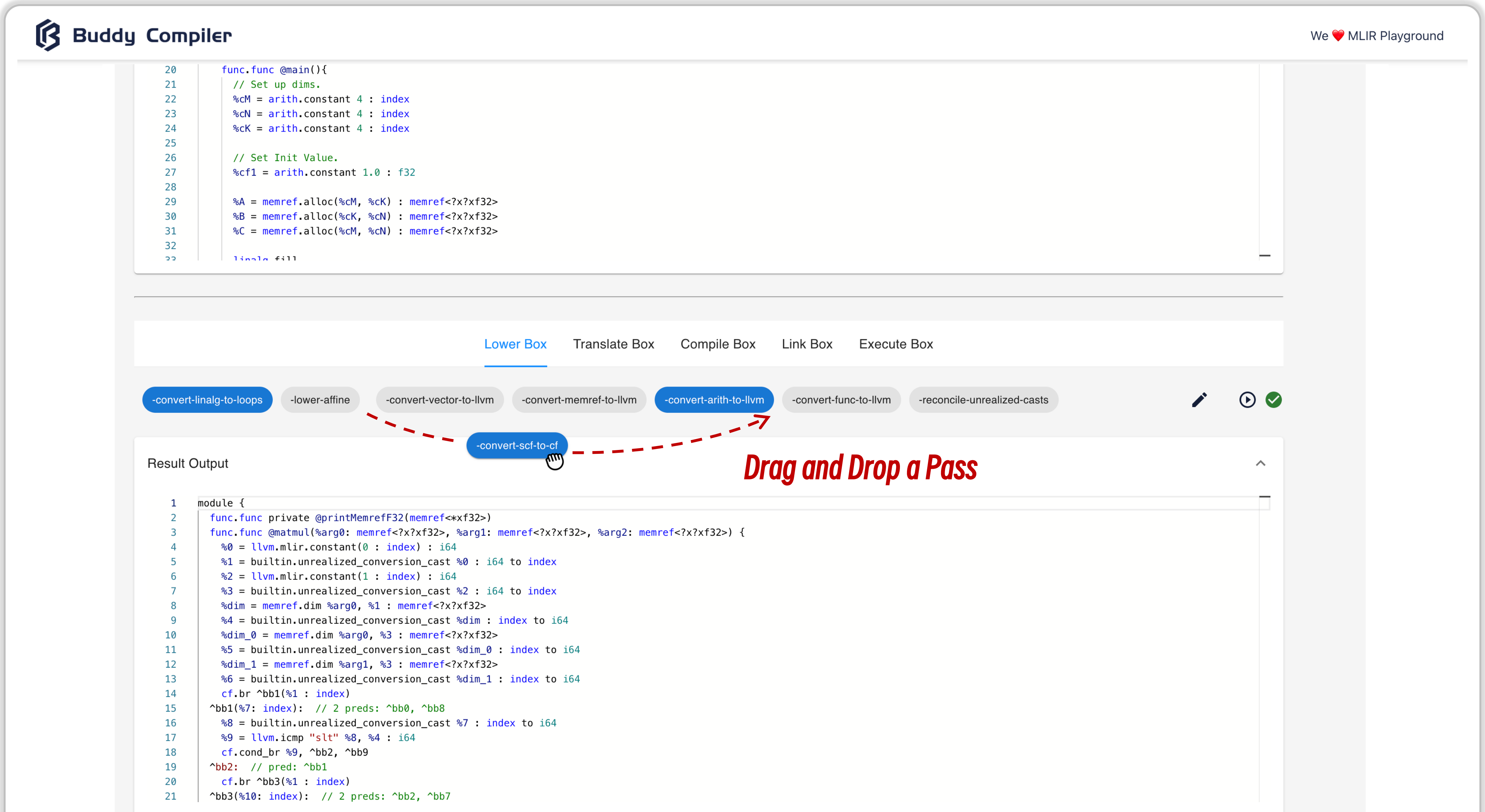
```
  func.func @main(){
    // Set up dims.
    %cM = arith.constant 4 : index
    %cN = arith.constant 4 : index
    %cK = arith.constant 4 : index

    // Set Init Value.
    %cf1 = arith.constant 1.0 : f32

    %A = memref.alloc(%cM, %cK) : memref<?x?xf32>
    %B = memref.alloc(%cK, %cN) : memref<?x?xf32>
    %C = memref.alloc(%cM, %cN) : memref<?x?xf32>

    linalg.fill
```

```
module {
  func.func private @printMemrefF32(memref<*xf32>)
  func.func @matmul(%arg0: memref<?x?xf32>, %arg1: memref<?x?xf32>, %arg2: memref<?x?xf32>) {
    %0 = llvm.mlir.constant(0 : index) : i64
    %1 = builtin.unrealized_conversion_cast %0 : i64 to index
    %2 = llvm.mlir.constant(1 : index) : i64
    %3 = builtin.unrealized_conversion_cast %2 : i64 to index
    %dim = memref.dim %arg0, %1 : memref<?x?xf32>
    %4 = builtin.unrealized_conversion_cast %dim : index to i64
    %dim_0 = memref.dim %arg0, %3 : memref<?x?xf32>
    %5 = builtin.unrealized_conversion_cast %dim_0 : index to i64
    %dim_1 = memref.dim %arg1, %3 : memref<?x?xf32>
    %6 = builtin.unrealized_conversion_cast %dim_1 : index to i64
    cf.br ^bb1(%1 : index)
  ^bb1(%7: index):  // 2 preds: ^bb0, ^bb8
    %8 = builtin.unrealized_conversion_cast %7 : index to i64
    %9 = llvm.icmp "slt" %8, %4 : i64
    cf.cond_br %9, ^bb2, ^bb9
  ^bb2:  // pred: ^bb1
    cf.br ^bb3(%1 : index)
  ^bb3(%10: index):  // 2 preds: ^bb2, ^bb7
```

Compiler As A Service for MLIR (https://buddy.isrc.ac.cn/)

Buddy Compiler

Buddy-CAAS serves MLIR users and developers to configure the pass pipeline and demonstrate on multiple backends quickly and smoothly.



```
func.func @main(){
  // Set up dims.
  %cM = arith.constant 4 : index
  %cN = arith.constant 4 : index
  %cK = arith.constant 4 : index

  // Set Init Value.
  %cf1 = arith.constant 1.0 : f32

  %A = memref.alloc(%cM, %cK) : memref<?x?xf32>
  %B = memref.alloc(%cK, %cN) : memref<?x?xf32>
  %C = memref.alloc(%cM, %cN) : memref<?x?xf32>

  linalg.fill
```

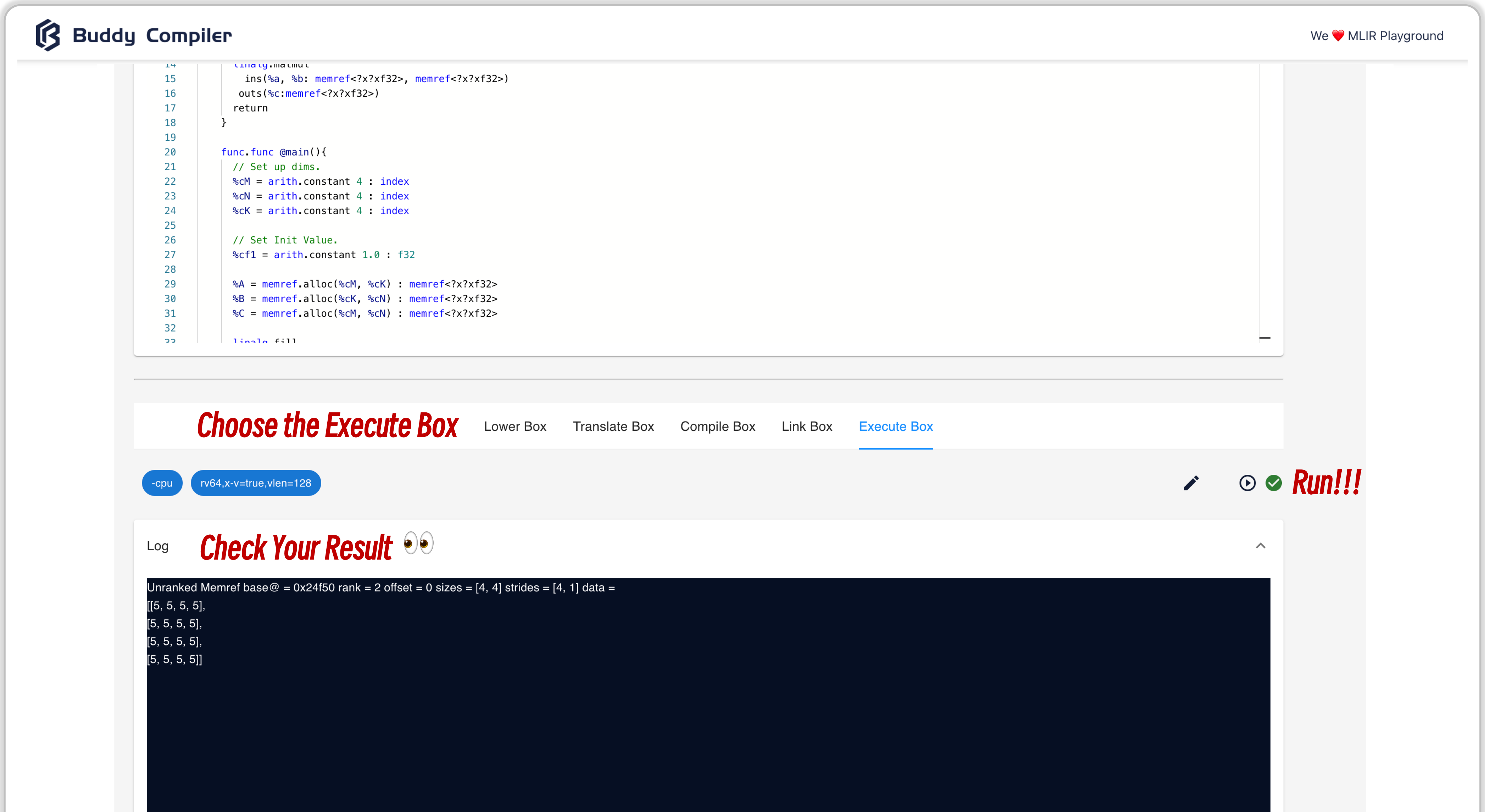
Result Output

```
module {
  func.func private @printMemrefF32(memref<*xf32>)
  func.func @matmul(%arg0: memref<?x?xf32>, %arg1: memref<?x?xf32>, %arg2: memref<?x?xf32>) {
    %0 = llvm.mlir.constant(0 : index) : i64
    %1 = builtin.unrealized_conversion_cast %0 : i64 to index
    %2 = llvm.mlir.constant(1 : index) : i64
    %3 = builtin.unrealized_conversion_cast %2 : i64 to index
    %dim = memref.dim %arg0, %1 : memref<?x?xf32>
    %4 = builtin.unrealized_conversion_cast %dim : index to i64
    %dim_0 = memref.dim %arg0, %3 : memref<?x?xf32>
    %5 = builtin.unrealized_conversion_cast %dim_0 : index to i64
    %dim_1 = memref.dim %arg1, %3 : memref<?x?xf32>
    %6 = builtin.unrealized_conversion_cast %dim_1 : index to i64
    cf.br ^bb1(%1 : index)
  ^bb1(%7: index):  // 2 preds: ^bb0, ^bb8
    %8 = builtin.unrealized_conversion_cast %7 : index to i64
    %9 = llvm.icmp "slt" %8, %4 : i64
    cf.cond_br %9, ^bb2, ^bb9
  ^bb2:  // pred: ^bb1
    cf.br ^bb3(%1 : index)
  ^bb3(%10: index):  // 2 preds: ^bb2, ^bb7
```

Compiler As A Service for MLIR (https://buddy.isrc.ac.cn/)

Buddy Compiler

Buddy-CAAS serves MLIR users and developers to configure the pass pipeline and demonstrate on multiple backends quickly and smoothly.



Buddy Compiler

We ❤️ MLIR Playground

```
    linalg.matmul
      ins(%a, %b: memref<?x?xf32>, memref<?x?xf32>)
     outs(%c:memref<?x?xf32>)
    return
  }

  func.func @main(){
    // Set up dims.
    %cM = arith.constant 4 : index
    %cN = arith.constant 4 : index
    %cK = arith.constant 4 : index

    // Set Init Value.
    %cf1 = arith.constant 1.0 : f32

    %A = memref.alloc(%cM, %cK) : memref<?x?xf32>
    %B = memref.alloc(%cK, %cN) : memref<?x?xf32>
    %C = memref.alloc(%cM, %cN) : memref<?x?xf32>

    linalg.fill
```

Choose the Execute Box

-cpu rv64,x-v=true,vlen=128

Run!!!

Log

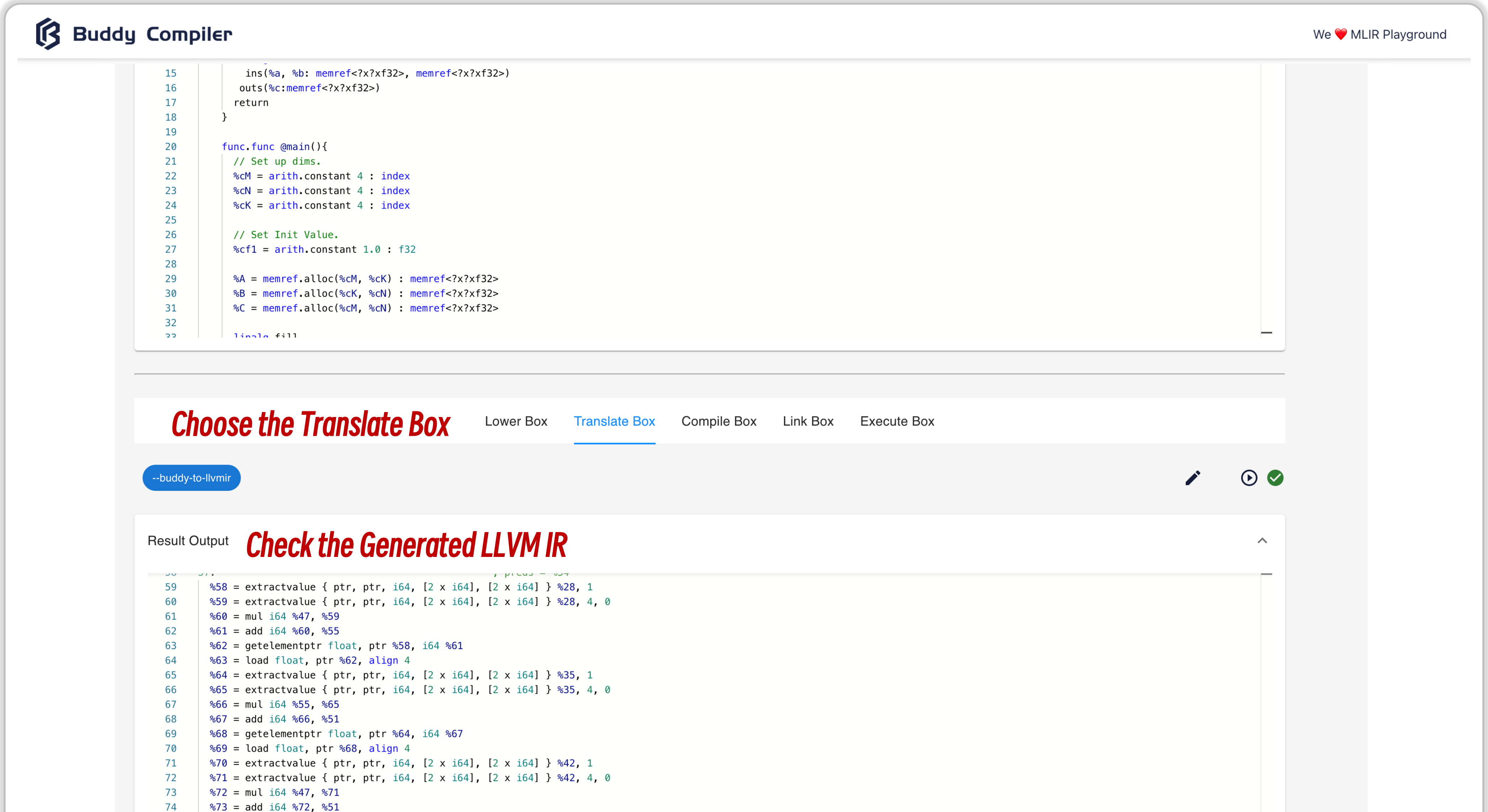
Check Your Result 👀

```
Unranked Memref base@ = 0x24f50 rank = 2 offset = 0 sizes = [4, 4] strides = [4, 1] data =
[[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5],
[5, 5, 5, 5]]
```

Compiler As A Service for MLIR (https://buddy.isrc.ac.cn/)

Buddy Compiler

Buddy-CAAS serves MLIR users and developers to configure the pass pipeline and demonstrate on multiple backends quickly and smoothly.



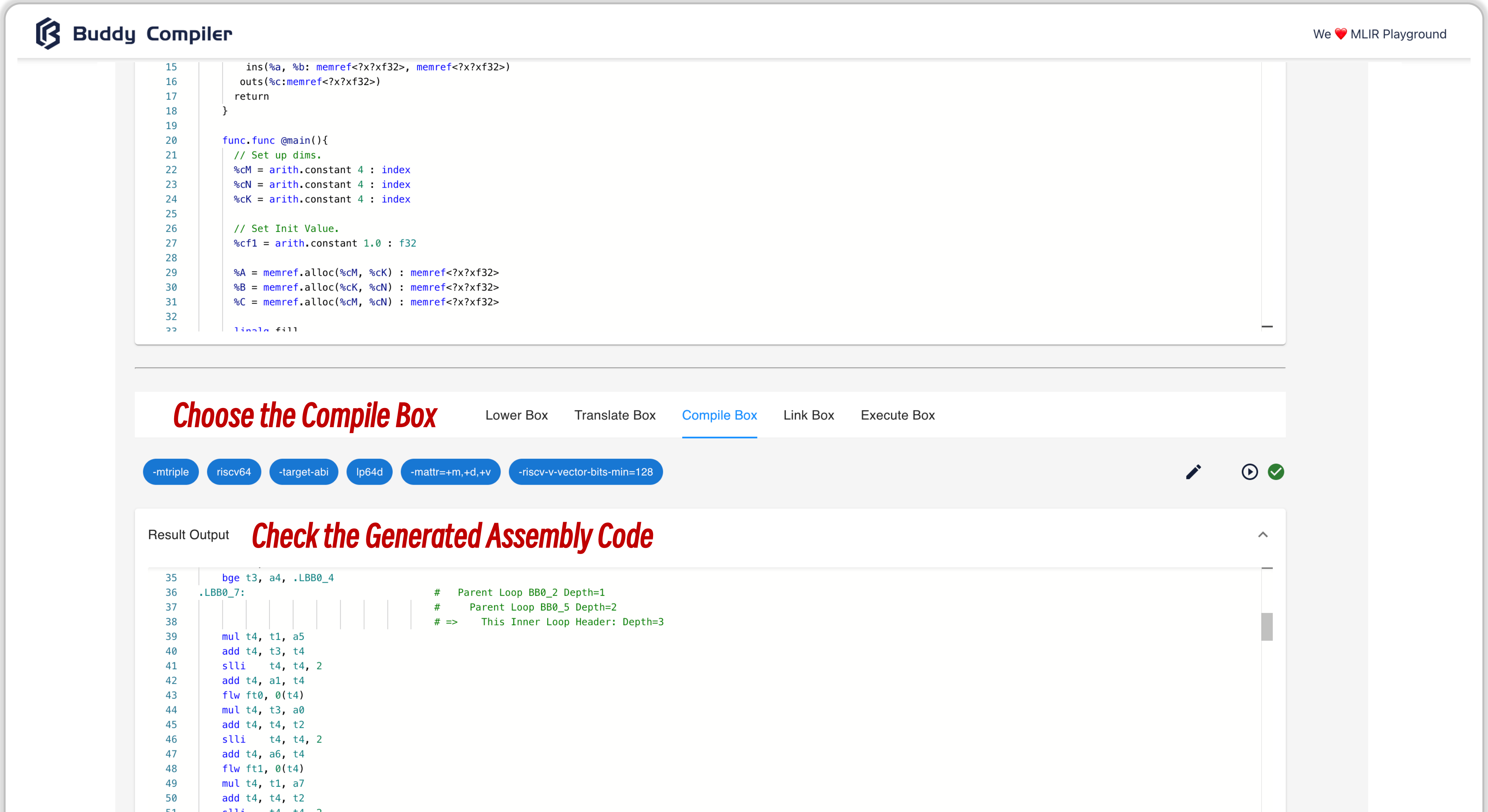
```
     ins(%a, %b: memref<?x?xf32>, memref<?x?xf32>)
    outs(%c:memref<?x?xf32>)
   return
 }

 func.func @main(){
   // Set up dims.
   %cM = arith.constant 4 : index
   %cN = arith.constant 4 : index
   %cK = arith.constant 4 : index

   // Set Init Value.
   %cf1 = arith.constant 1.0 : f32

   %A = memref.alloc(%cM, %cK) : memref<?x?xf32>
   %B = memref.alloc(%cK, %cN) : memref<?x?xf32>
   %C = memref.alloc(%cM, %cN) : memref<?x?xf32>
```

```
  %58 = extractvalue { ptr, ptr, i64, [2 x i64], [2 x i64] } %28, 1
  %59 = extractvalue { ptr, ptr, i64, [2 x i64], [2 x i64] } %28, 4, 0
  %60 = mul i64 %47, %59
  %61 = add i64 %60, %55
  %62 = getelementptr float, ptr %58, i64 %61
  %63 = load float, ptr %62, align 4
  %64 = extractvalue { ptr, ptr, i64, [2 x i64], [2 x i64] } %35, 1
  %65 = extractvalue { ptr, ptr, i64, [2 x i64], [2 x i64] } %35, 4, 0
  %66 = mul i64 %55, %65
  %67 = add i64 %66, %51
  %68 = getelementptr float, ptr %64, i64 %67
  %69 = load float, ptr %68, align 4
  %70 = extractvalue { ptr, ptr, i64, [2 x i64], [2 x i64] } %42, 1
  %71 = extractvalue { ptr, ptr, i64, [2 x i64], [2 x i64] } %42, 4, 0
  %72 = mul i64 %47, %71
  %73 = add i64 %72, %51
```

Compiler As A Service for MLIR (https://buddy.isrc.ac.cn/)

Buddy Compiler

Buddy-CAAS serves MLIR users and developers to configure the pass pipeline and demonstrate on multiple backends quickly and smoothly.



```
      ins(%a, %b: memref<?x?xf32>, memref<?x?xf32>)
     outs(%c:memref<?x?xf32>)
    return
  }

  func.func @main(){
    // Set up dims.
    %cM = arith.constant 4 : index
    %cN = arith.constant 4 : index
    %cK = arith.constant 4 : index

    // Set Init Value.
    %cf1 = arith.constant 1.0 : f32

    %A = memref.alloc(%cM, %cK) : memref<?x?xf32>
    %B = memref.alloc(%cK, %cN) : memref<?x?xf32>
    %C = memref.alloc(%cM, %cN) : memref<?x?xf32>

    linalg.fill
```

Choose the Compile Box

-mtriple riscv64 -target-abi lp64d -mattr=+m,+d,+v -riscv-v-vector-bits-min=128

Result Output **Check the Generated Assembly Code**

```
	bge	t3, a4, .LBB0_4
.LBB0_7:                                #   Parent Loop BB0_2 Depth=1
                                        #     Parent Loop BB0_5 Depth=2
                                        # =>    This Inner Loop Header: Depth=3
	mul	t4, t1, a5
	add	t4, t3, t4
	slli	t4, t4, 2
	add	t4, a1, t4
	flw	ft0, 0(t4)
	mul	t4, t3, a0
	add	t4, t4, t2
	slli	t4, t4, 2
	add	t4, a6, t4
	flw	ft1, 0(t4)
	mul	t4, t1, a7
	add	t4, t4, t2
	slli	t4, t4, 2
```

Buddy Compiler

Thanks

https://buddy.isrc.ac.cn/