Seal5 full flow

Fresh run with integrated PatternGen; optional controls are shown as annotations.

01 Initialize and load

Clone / select LLVM; create metadata
Load CoreDSL, YAML configuration and tests

02 Setup and bootstrap patch

Fetch CoreDSL2LLVM dependency
Apply PHASE_0 support patches

03 Initial LLVM build

Configure and compile the bootstrapped toolchain
Skipped when reusing an existing build

04 Transform models

T01–T38: convert, normalize, infer and annotate
Create annotated Seal5 models + full CoreDSL

05 Generate structural artifacts

G01–G11: instruction / target / frontend metadata
Create patch indices and generated tests

06 Patch and rebuild

Apply PHASE_1 and PHASE_2
Build the instruction-aware LLVM toolchain

07 Build extraction tools

Build pattern-gen and llc
Required before instruction-selection rule extraction

08 Generate selection patterns

G12: P01–P04 (see detail at right)
Produce TableGen patterns or direct GISel matchers

09 Apply remaining phases

Apply PHASE_3 → PHASE_4 → PHASE_5
PHASE_4 receives the generated selection rules

10 Final LLVM build

Compile the retargeted compiler
Includes the generated instruction-selection rules

11 Validate and deliver

Run imported + generated llvm-lit tests
Optionally install, deploy sources, export metadata

Inputs

CoreDSL architecture / instruction descriptions
YAML: filters, extensions, passes and build settings
C / LLVM IR / assembly test files

Model artifacts

CoreDSL → M2-ISA-R model → Seal5 model
Instruction sets, operands, behavior and attributes
Normalized behavior + per-model settings

Generated integration

TableGen definitions; ISA / feature metadata
Intrinsics / builtins; disassembler integration
Patch indices + generated test sources

PatternGen detail · G12

`P01 write_cdsl_compat`
Reduced CoreDSL, split per instruction

`P02 behav_to_llvmir`
LLVM IR inspection artifact (.ll)

`P03 llvmir_to_gmir`
Generic MIR inspection artifact (.gmir)

`P04 behav_to_pat`
pattern-gen consumes reduced CoreDSL again

Internally: behavior → LLVM IR → GMIR → matcher
Output: .td patterns; optional direct .inc matchers
P02 / P03 files are not direct inputs to P04.

Outputs

Retargeted LLVM / Clang / assembler tools
Test results, metrics and generated artifacts
Optional installation and source / metadata archives

Execution controls: PREPATCHED reuses phase 0; RERUN suppresses the initial / intermediate all-target builds.
SKIP_PATTERNS omits extraction tools, G12 and phases 3–5. TEST_ONLY bypasses setup, generation, patching and builds.
Lifecycle flags can disable individual operations; reset and final cleanup are optional. GNU patch generation is opt-in.
Solid arrows: main execution / artifact relationships. Dashed arrows inside G12: pass order, not artifact dependencies.
Source: seal5/wrapper.py · seal5/flow.py · seal5/pass_list.py