Case study of Optimized CXL platforms for AI/ML Check Points

Presenter:

Sathish Kumar M

Associate Technical Director

Samsung Semiconductor Inc

Ratish Gopinath

Associate Staff Engineer

Samsung Semiconductor Inc

Arun V Pillai

Senior Staff Engineer

Samsung Semiconductor Inc

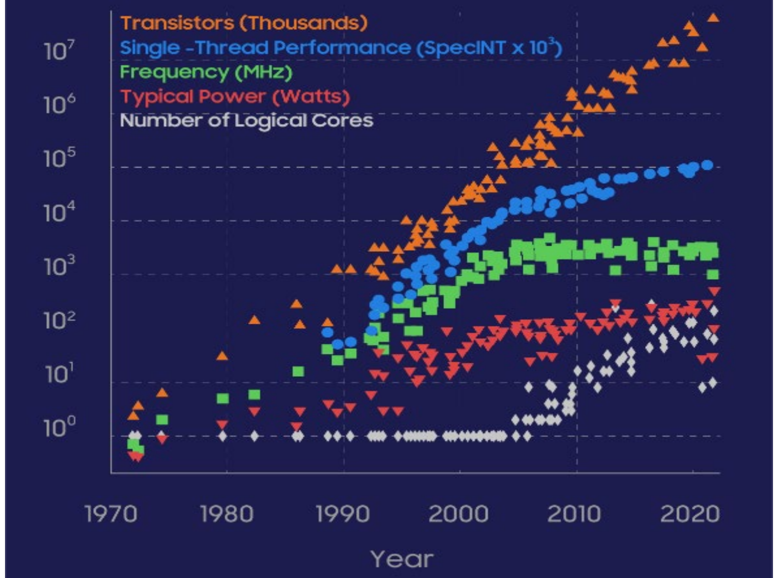
Agenda

- Background
- CXL and use cases
- CXL Ecosystem in AI/ML Infra
- CXL Direct access
- CNN Ecosystem Comparison
- Case study Results
- Summary

Data Growth vs Processor Scaling

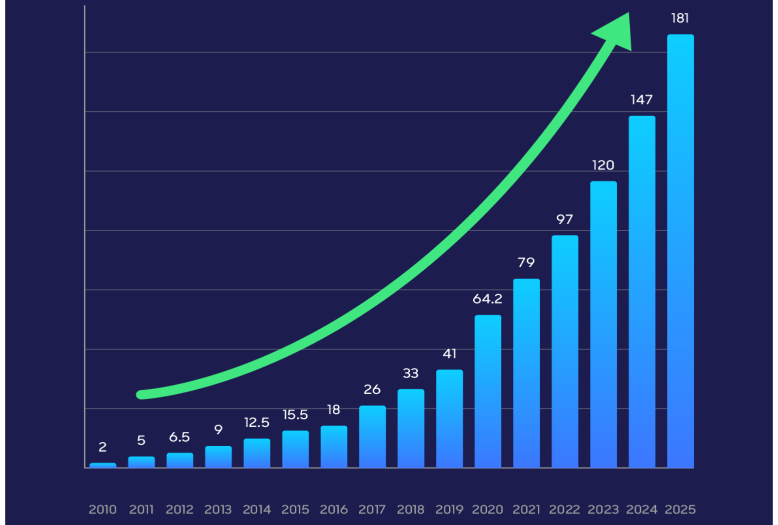
CPU Performance Growth is slowing down

50 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F, Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

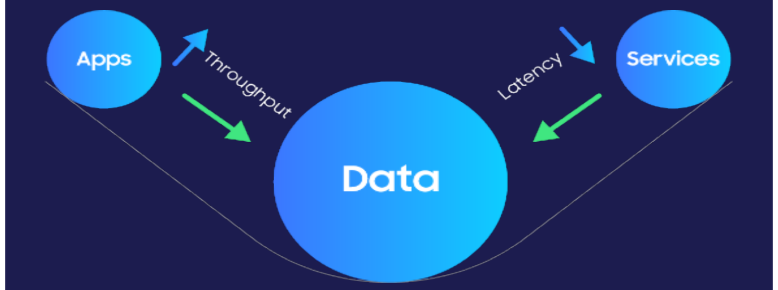
Volume of Data exponentially increases



Source: Statista

Data Gravity

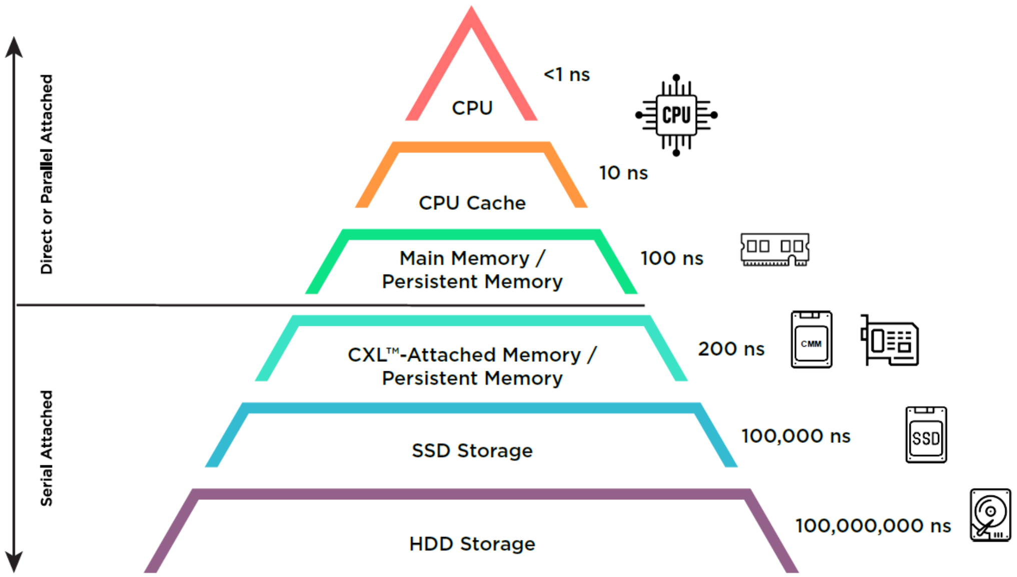
Moving data closer to compute

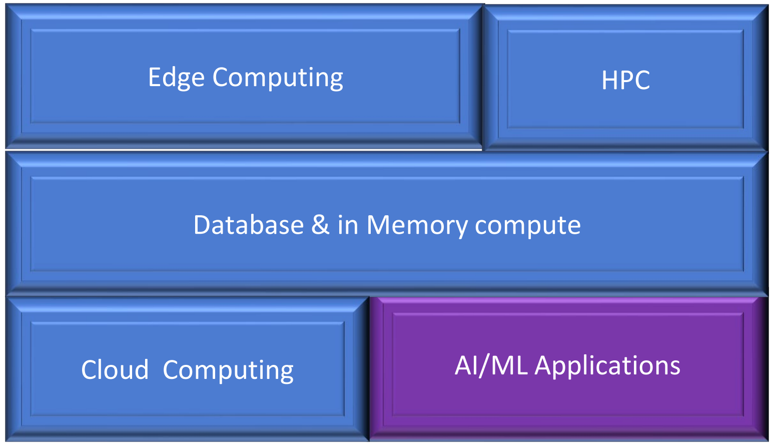


Source: Medium

CXL Industry Trends

- CPU to reach data much faster and closer
  - 500x faster
- Many Industry usecases

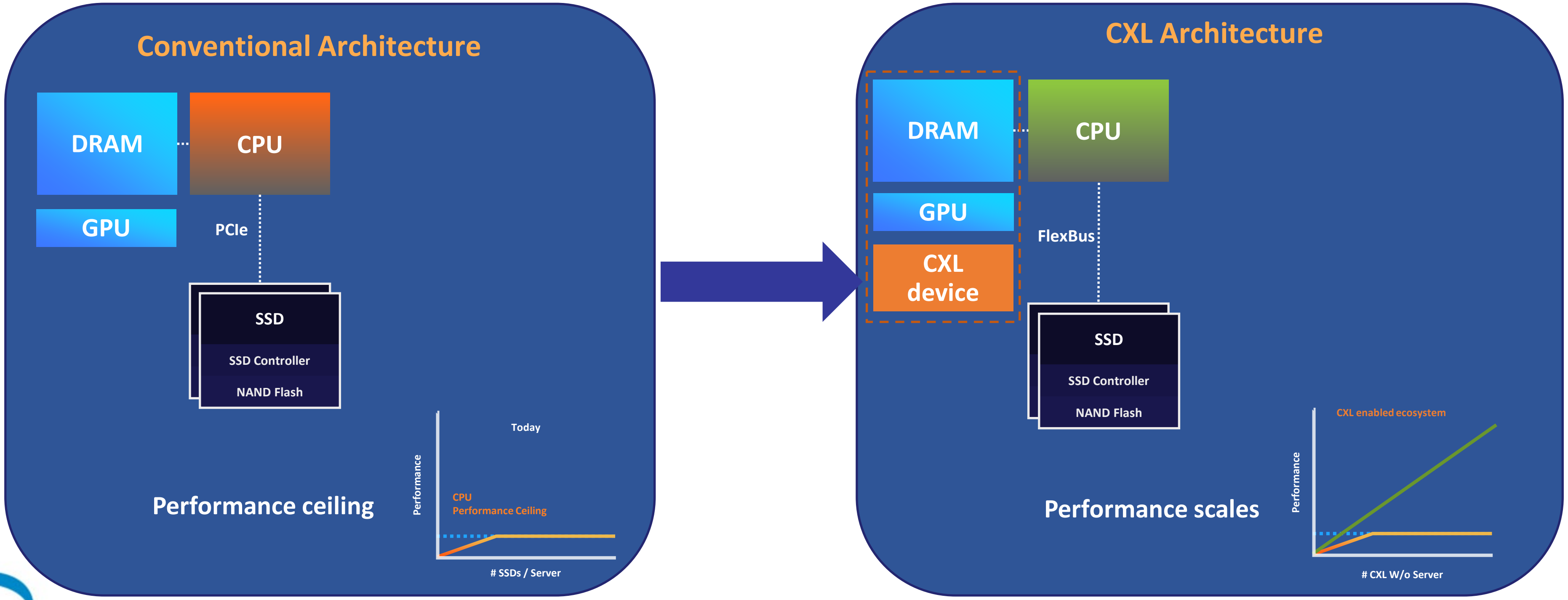




CXL Use cases

CXL Ecosystem in AI/ML Infrastructure

- Conventional infrastructure:
  - CPU + GPU + Memory + SSD
  - Memory is limited
  - Long CPU waiting for read/write
- CXL infrastructure:
  - CPU + GPU + Memory + CXL memory devices + SSD
  - CXL brings
    - More memory
    - Dynamic expansion



Conventional Architecture

DRAM

CPU

GPU

PCIe

SSD

SSD Controller

NAND Flash

Performance ceiling

CXL Architecture

DRAM

CPU

GPU

FlexBus

CXL device

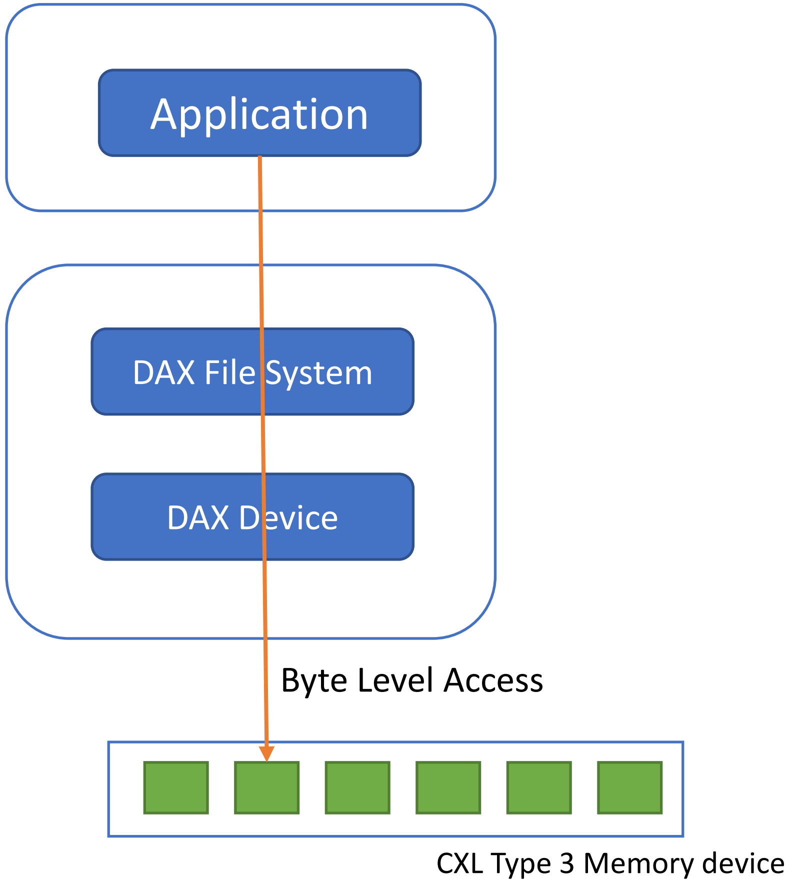
SSD

SSD Controller

NAND Flash

Performance scales

CXL - DAX access



DAX(Direct Access) File System:

- Memory is byte oriented
- Byte level access
  - CXL memory/Persistent memory
- Dual benefit
  - CXL + DAX

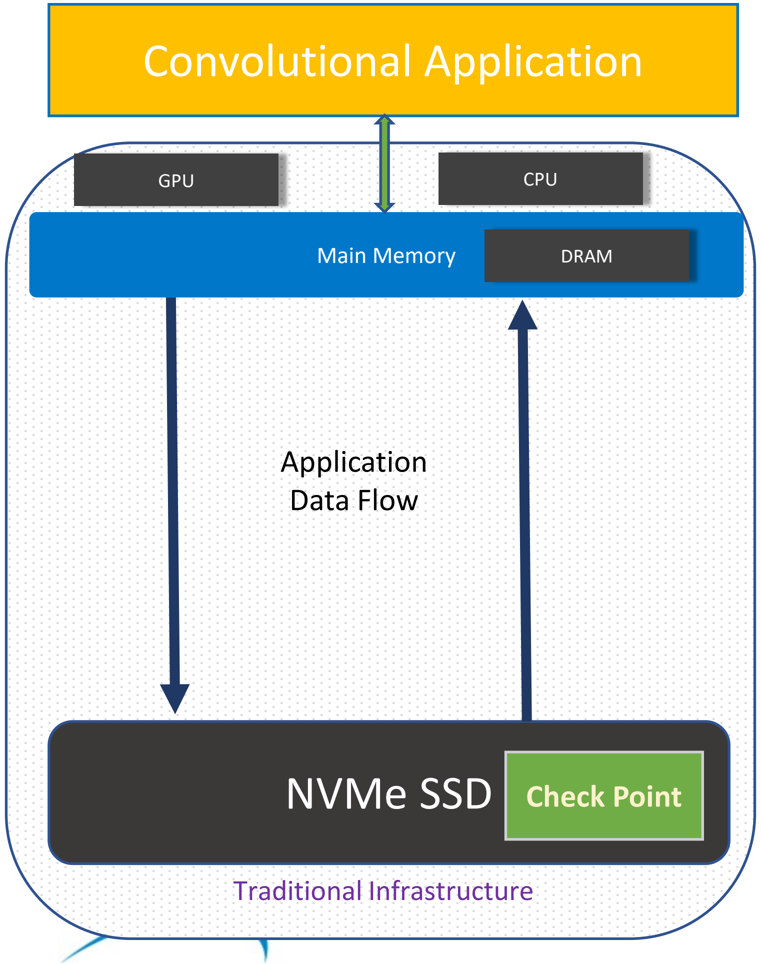
CXL Memory provisioning:

1. Create namespace using ‘**ndctl’** on the CXL memory
2. Create xfs filesystem
3. Mount with ‘-o dax’ option

Applications can use mounted DAX file system directly

CNN Ecosystem Comparison

- U-Net: Convolutional Neural Networks for Biomedical Image Segmentation
  - Brain tumor image segmentation for the MRI 3D-scan images
  - High In-Memory computing and large checkpoints
  - Focus on checkpoints on the low latency device



Convolutional Application

GPU

CPU

Main Memory

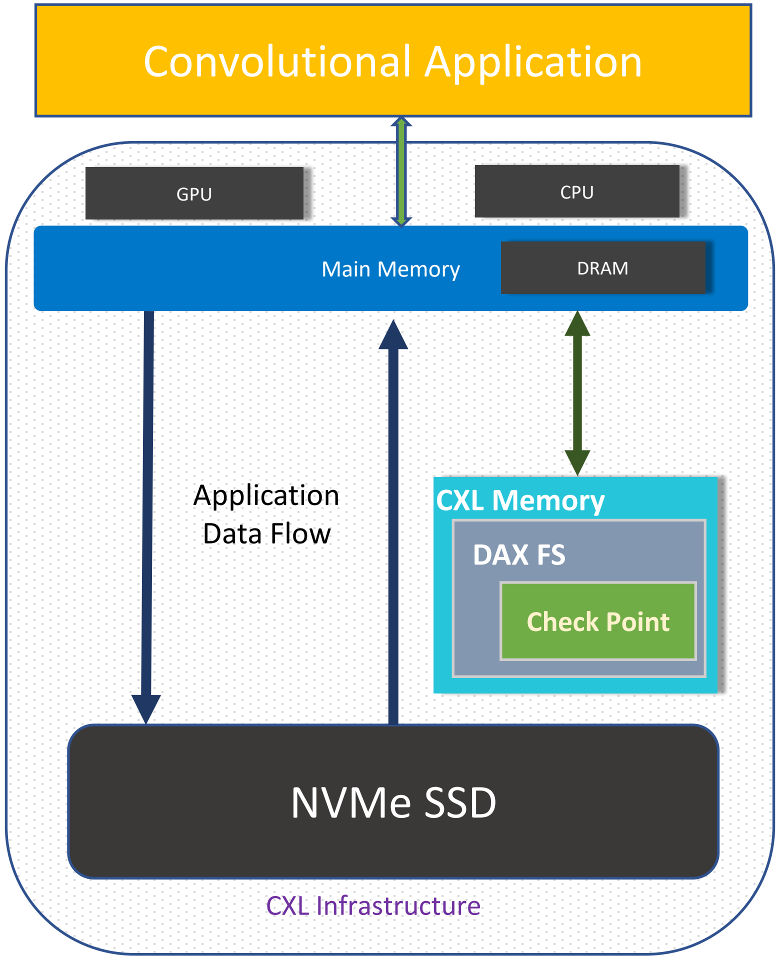
DRAM

Application Data Flow

NVMe SSD

Check Point

Traditional Infrastructure



Convolutional Application

GPU

CPU

Main Memory

DRAM

Application Data Flow

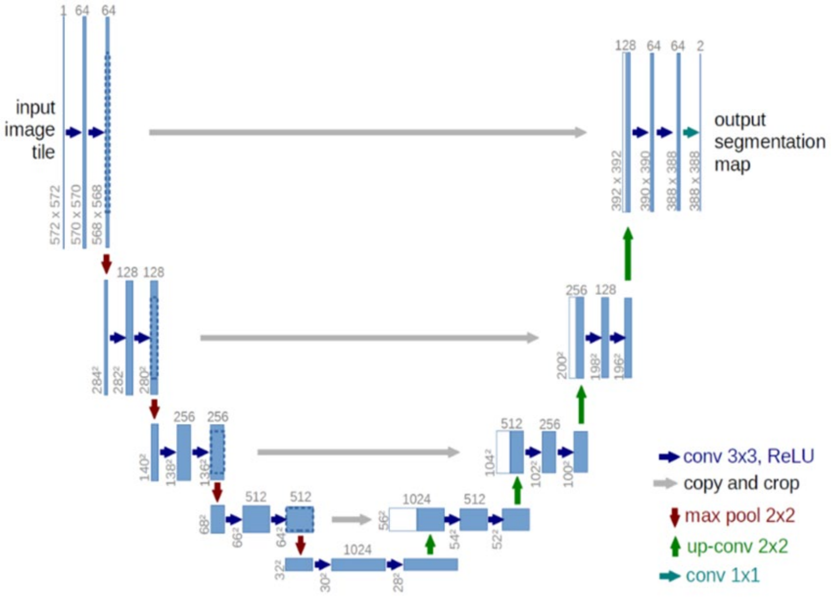
CXL Memory

DAX FS

Check Point

NVMe SSD

CXL Infrastructure

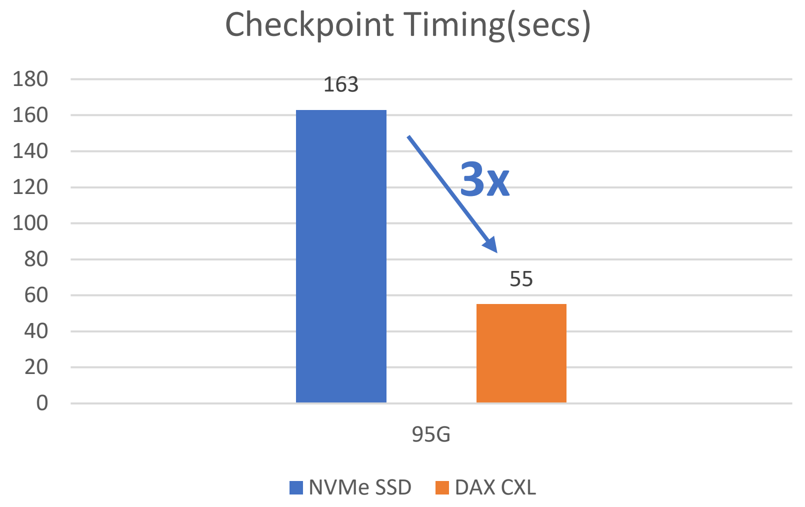


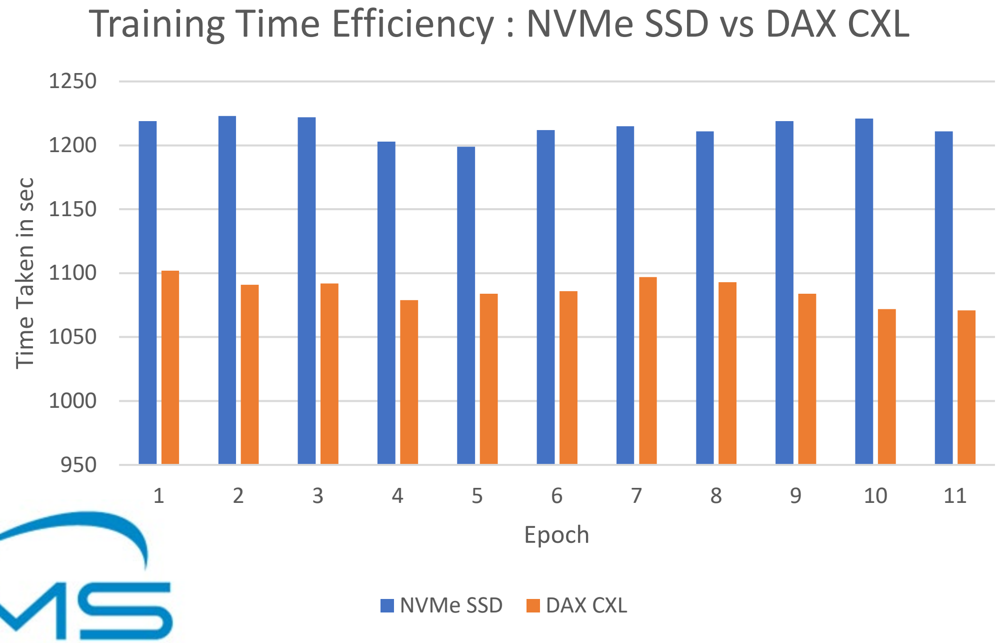
Source : https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

Check Points(CP)

- Safeguards intermediate models
- CP size increases with complexity

Study Results





Sample data Set details	
Image Size	32 x 32
Volume slices taken from each image	3 [63-66]
Scale factor	32
Dataset size	13GB

~20% latency reduction on training time

Summary

- Saving Checkpoint in CXL memory reduces the training time
- Disaggregated CXL infra helps to reduce the TCO
  - Memory will be pooled across AI/ML infrastructure
  - CXL memory can be reused across systems and applications
  - Memory infra can be scaled in terms of capacity and bandwidth

<u>Future work:</u>

- Experiment the checkpoint latency with LLM applications

Enhancement of Functional code Coverage in UVM-based verification environment using LLM-based model

Ramya B T

Senior Staff Engineer

Samsung Semiconductor Inc

Harshit Sharma

Student Trainee

Samsung Semiconductor Inc

Sairam Jujjarapu

Associate Staff Engineer

Samsung Semiconductor Inc

Sachin Suresh Upadhya

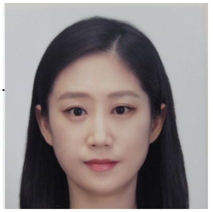
Senior Staff Engineer

Samsung Semiconductor Inc

Keerthi Kiran J

Director

Samsung Semiconductor Inc



Kyungmin Kim

Principal Engineer

Samsung Electronics

Sathish Kumar M

Associate Technical Director

Samsung Semiconductor Inc

Agenda

- Background and Purpose
- Proposed Solution
- Architecture
- Experimental setup
- Execution and Results
- Conclusion

Background and Purpose

- UVM (Universal Verification Methodology)
  - Standardized methodology for verifying digital designs
  - Provides a mechanism for building functional test benches for achieving 100% functionality coverage

- Types of testing
  - Random testing
  - Directed testing
  - Coverage-driven testing

- Drawbacks: Though randomization is supported in UVM,
  - It takes time to achieve a 100% coverage
  - Manual effort is needed to identify coverage holes and write directed tests to cover them

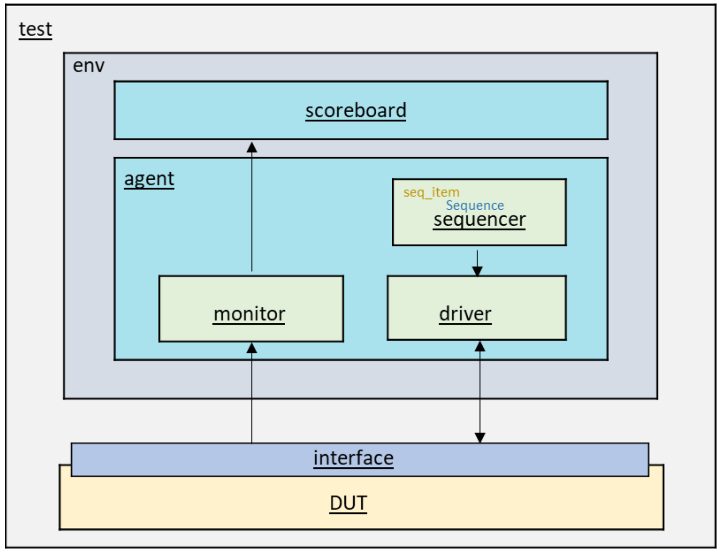


Fig : UVM architecture

- ML-based tools can be good for this purpose to reduce manual effort and achieve high coverage

Proposed Solution

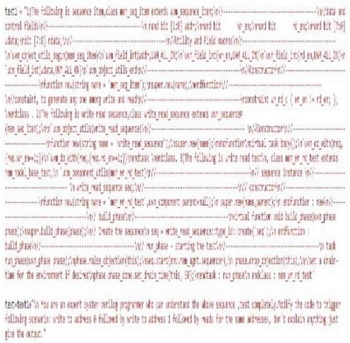
- Python-based script shall generate the prompt for the model
  - Prompt shall be specific and exhaustive enough to enable the model to generate the tests

- Pre-trained LLM model
  - Model used shall be pre-trained with system verilog syntaxes and UVM rules/keywords and UVM TB building
  - Automates the process of test generation in UVM environment
  - Target to achieve higher code coverage by generating sequences
  - Deepseek LLM is chosen for this solution. Model is pre-trained to learn the UVM sequences and testcase generation mechanism

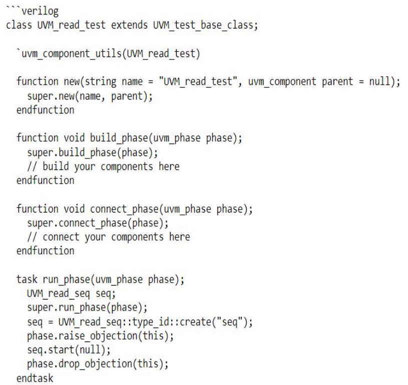
- The generated test shall be run on DUT (Device Under Test) with UVM for coverage

- The coverage holes after UVM randomization can be aimed to be covered with the proposed model

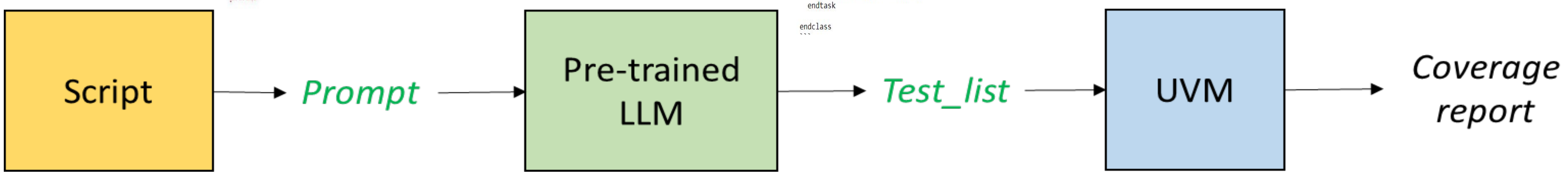
Architecture of proposed solution

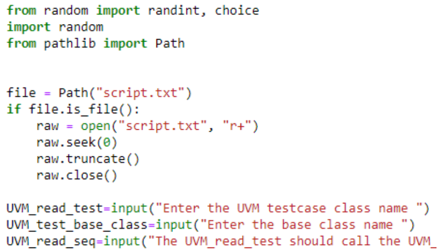


```
text1 = "1)The following is sequence item,class mem_seq_item extends uvm_sequence_item;\n//--------------------------------------------\n//data and
control fields\n//------------------------------------------\n rand bit [1:0] addr;\nrand bit       wr_en;\nrand bit       rd_en;\nrand bit [7:0]
wdata;\nbit [7:0] rdata;\n//---------------------------------------------\n//Utility and Field macros\n//-------------------------------------------
\n`uvm_object_utils_begin(mem_seq_item)\n`uvm_field_int(addr,UVM_ALL_ON)\n`uvm_field_int(wr_en,UVM_ALL_ON)\n`uvm_field_int(rd_en,UVM_ALL_ON)\n
`uvm_field_int(wdata,UVM_ALL_ON)\n`uvm_object_utils_end\n//-------------------------------------------\n//Constructor\n//----------------------
-------------\nfunction new(string name = "mem_seq_item");\nsuper.new(name);\nendfunction\n//-------------------------------------------
\n//constaint, to generate any one among write and read\n//--------------------------------------------\nconstraint wr_rd_c { wr_en != rd_en; };
\nendclass . 2)The following is write read sequence,class write_read_sequence extends uvm_sequence#
(mem_seq_item);\n\n`uvm_object_utils(write_read_sequence)\n//------------------------------------------ \n//Constructor\n//-------------------
-----------------\nfunction new(string name = "write_read_sequence");\nsuper.new(name);\nendfunction\n\nvirtual task body();\n`uvm_do_with(req,
{req.wr_en==1;})\n`uvm_do_with(req,{req.rd_en==1;})\nendtask \nendclass. 3)The following is write read test\n, class mem_wr_rd_test extends
mem_model_base_test;\n `uvm_component_utils(mem_wr_rd_test)\n//-----------------------------------------\n// sequence instance \n//-----------
------------------------- \n write_read_sequence seq;\n//-----------------------------------------\n// constructor\n//----------------------
-------------\nfunction new(string name = "mem_wr_rd_test",uvm_component parent=null);\n super.new(name,parent);\n endfunction : new\n//-----
---------------------------------\n// build_phase\n//------------------------------------------\nvirtual function void build_phase(uvm_phase
phase);\nsuper.build_phase(phase);\n\n// Create the sequence\n seq = write_read_sequence::type_id::create("seq");\n endfunction :
build_phase\n//------------------------------------------\n// run_phase - starting the test\n//------------------------------------------\n task
run_phase(uvm_phase phase);\nphase.raise_objection(this);\nseq.start(env.mem_agnt.sequencer);\n phase.drop_objection(this);\n\n//set a drain-
time for the environment if desired\nphase.phase_done.set_drain_time(this, 50);\nendtask : run_phase\n endclass : mem_wr_rd_test"

text=text1+"\n You are an expert system verilog programmer who can understand the above sequence ,test completely.Modify the code to trigger
following scenario: write to address 0 followed by write to address 1 followed by reads for the same addresses, don't explain anything just
give the output."
```



~~~
```verilog
class UVM_read_test extends UVM_test_base_class;

  `uvm_component_utils(UVM_read_test)

  function new(string name = "UVM_read_test", uvm_component parent = null);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    // build your components here
  endfunction

  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    // connect your components here
  endfunction

  task run_phase(uvm_phase phase);
    UVM_read_seq seq;
    super.run_phase(phase);
    seq = UVM_read_seq::type_id::create("seq");
    phase.raise_objection(this);
    seq.start(null);
    phase.drop_objection(this);
  endtask

endclass
```
~~~



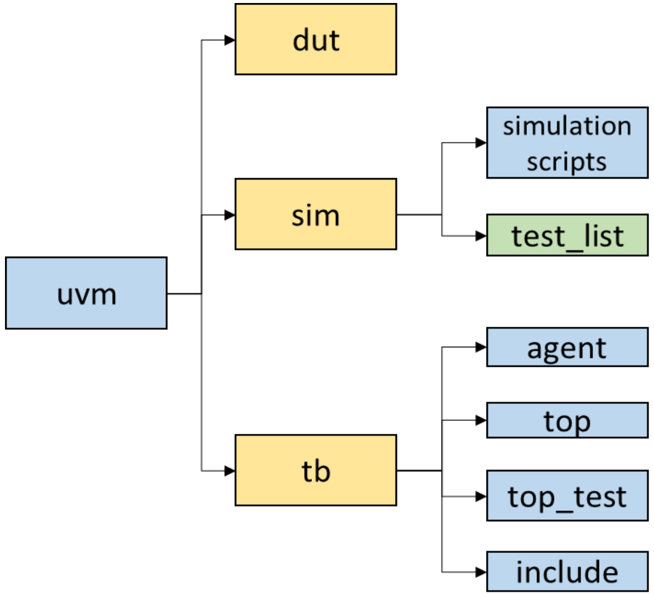


```
from random import randint, choice
import random
from pathlib import Path


file = Path("script.txt")
if file.is_file():
    raw = open("script.txt", "r+")
    raw.seek(0)
    raw.truncate()
    raw.close()

UVM_read_test=input("Enter the UVM testcase class name ")
UVM_test_base_class=input("Enter the base class name ")
UVM_read_seq=input("The UVM_read_test should call the UVM_
```

- Script → Python-based script to generate the prompt for LLM
- Prompt → Instructions for LLM with minimal details for test_list
- Pre-trained LLM → Deepseek model used to generate the test_list
- Test_list → Test with args
- UVM → Methodology to execute the test and generate coverage report

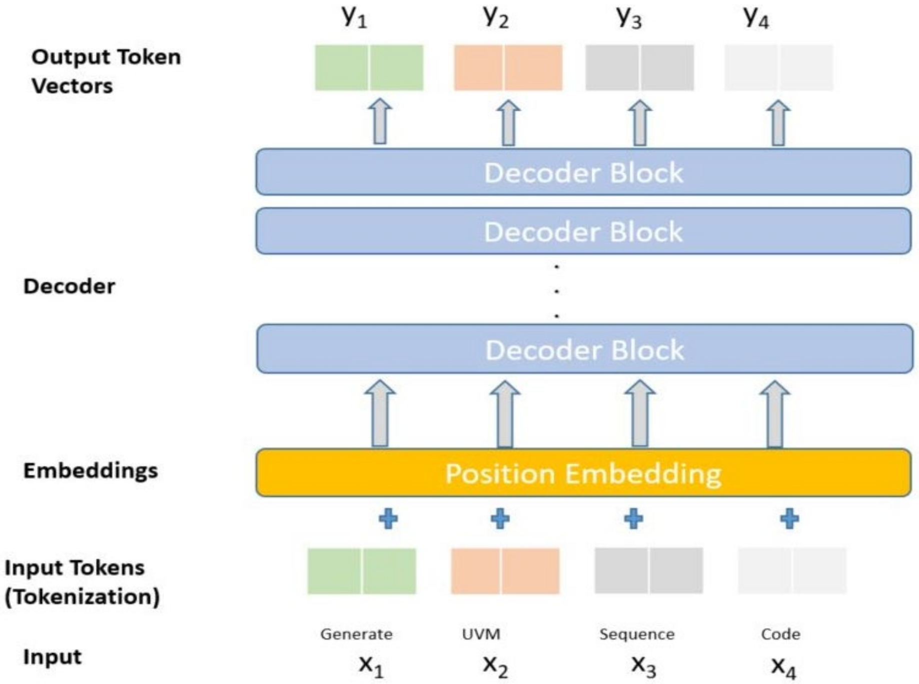
Functional block diagram

- **UVM**



UVM testbench has the test_list that combines the regression tests with which majority of coverage can be achieved

- **Pre-trained LLM**



x_i is the input code snippet and y_i is generated output code

Each decoder block contains masked self attention, Layer normalization and fully connected layer

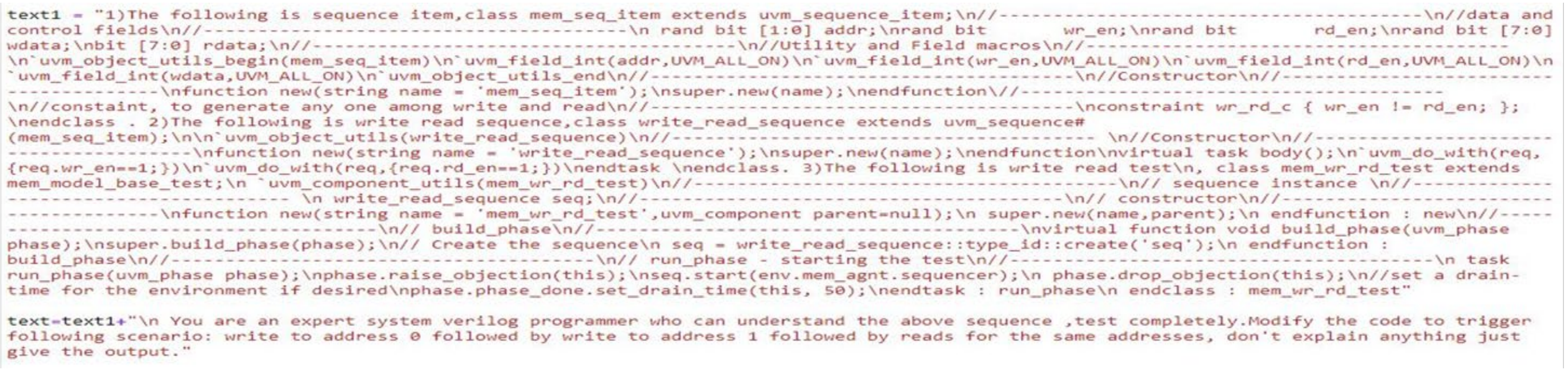
Experimental setup

Sl No	Item	Configuration
1	DUT	NAND Behavioral model
2	TB	Basic UVM testbench
3	Tool for test execution	xrun
4	LLM	Deepseek-coder-6.7b-instruct Reference : https://huggingface.co/deepseek-ai/deepseek-coder-6.7b-instruct

Execution and Results : Example 1

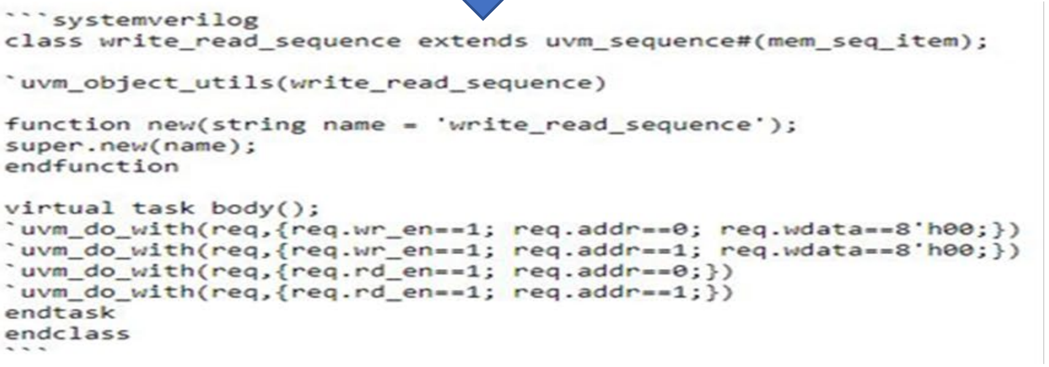
- we gave the code of mem_seq_item, write_read _sequence,and mem_write_read_test as input and asked the model to trigger the scenario in which write to address 0 is followed by write to address 1 followed by reads for the same address and the model is able to understand all the input code and the instructions and able to update read_write_sequence by creating new read/write sequences with specified address and random input data to write.

Instruction



```
text1 = "1)The following is sequence item,class mem_seq_item extends uvm_sequence_item;\n//------------------------------------\n//data and
control fields\n//-----------------------------------\n rand bit [1:0] addr;\nrand bit        wr_en;\nrand bit        rd_en;\nrand bit [7:0]
wdata;\nbit [7:0] rdata;\n//----------------------------------------\n//Utility and Field macros\n//-------------------------------------------
\n`uvm_object_utils_begin(mem_seq_item)\n`uvm_field_int(addr,UVM_ALL_ON)\n`uvm_field_int(wr_en,UVM_ALL_ON)\n`uvm_field_int(rd_en,UVM_ALL_ON)\n
`uvm_field_int(wdata,UVM_ALL_ON)\n`uvm_object_utils_end\n//---------------------------------------------\n//Constructor\n//---------------------------
--------------\nfunction new(string name = 'mem_seq_item');\nsuper.new(name);\nendfunction\//---------------------------------------
\n//constaint, to generate any one among write and read\n//----------------------------------------------\nconstraint wr_rd_c { wr_en != rd_en; };
\nendclass . 2)The following is write read sequence,class write_read_sequence extends uvm_sequence#
(mem_seq_item);\n\n`uvm_object_utils(write_read_sequence)\n//---------------------------------------- \n//Constructor\n//-------------------------
-----------------\nfunction new(string name = 'write_read_sequence');\nsuper.new(name);\nendfunction\nvirtual task body();\n`uvm_do_with(req,
{req.wr_en==1;})\n`uvm_do_with(req,{req.rd_en==1;})\nendtask \nendclass. 3)The following is write read test\n, class mem_wr_rd_test extends
mem_model_base_test;\n `uvm_component_utils(mem_wr_rd_test)\n//------------------------------------------\n// sequence instance \n//-------------
------------------------ \n write_read_sequence seq;\n//-----------------------------------------\n// constructor\n//-------------------------
--------------\nfunction new(string name = 'mem_wr_rd_test',uvm_component parent=null);\n super.new(name,parent);\n endfunction : new\n//-----
-----------------------------------\n// build_phase\n//---------------------------------\nvirtual function void build_phase(uvm_phase
phase);\nsuper.build_phase(phase);\n// Create the sequence\n seq = write_read_sequence::type_id::create('seq');\n endfunction :
build_phase\n//-------------------------------------------\n// run_phase - starting the test\n//-----------------------------------------\n task
run_phase(uvm_phase phase);\nphase.raise_objection(this);\nseq.start(env.mem_agnt.sequencer);\n phase.drop_objection(this);\n//set a drain-
time for the environment if desired\nphase.phase_done.set_drain_time(this, 50);\nendtask : run_phase\n endclass : mem_wr_rd_test"

text=text1+"\n You are an expert system verilog programmer who can understand the above sequence ,test completely.Modify the code to trigger
following scenario: write to address 0 followed by write to address 1 followed by reads for the same addresses, don't explain anything just
give the output."
```

Prompt



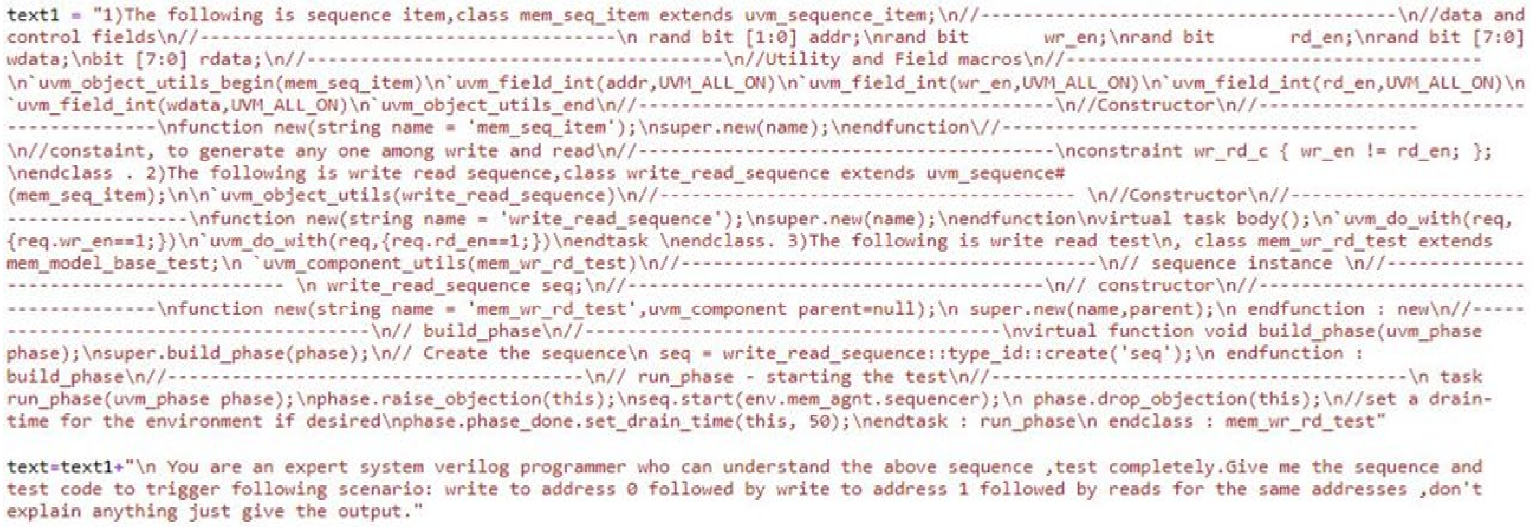
~~~
```systemverilog
class write_read_sequence extends uvm_sequence#(mem_seq_item);

`uvm_object_utils(write_read_sequence)

function new(string name = 'write_read_sequence');
super.new(name);
endfunction

virtual task body();
`uvm_do_with(req,{req.wr_en==1; req.addr==0; req.wdata==8'h00;})
`uvm_do_with(req,{req.wr_en==1; req.addr==1; req.wdata==8'h00;})
`uvm_do_with(req,{req.rd_en==1; req.addr==0;})
`uvm_do_with(req,{req.rd_en==1; req.addr==1;})
endtask
endclass
```
~~~

Output sequence

Execution and Results : Example 2 (1/2)

- we gave the code of mem_seq_item, write_read _sequence,and mem_write_read_test as input and asked the model to give the sequence and test code trigger the scenario in which write to address 0 is followed by write to address 1 followed by reads for the same address and the model is able to understand all the input code and the instructions and able to update read_write_sequence by creating new read/write sequences with specified address and random input data to write.

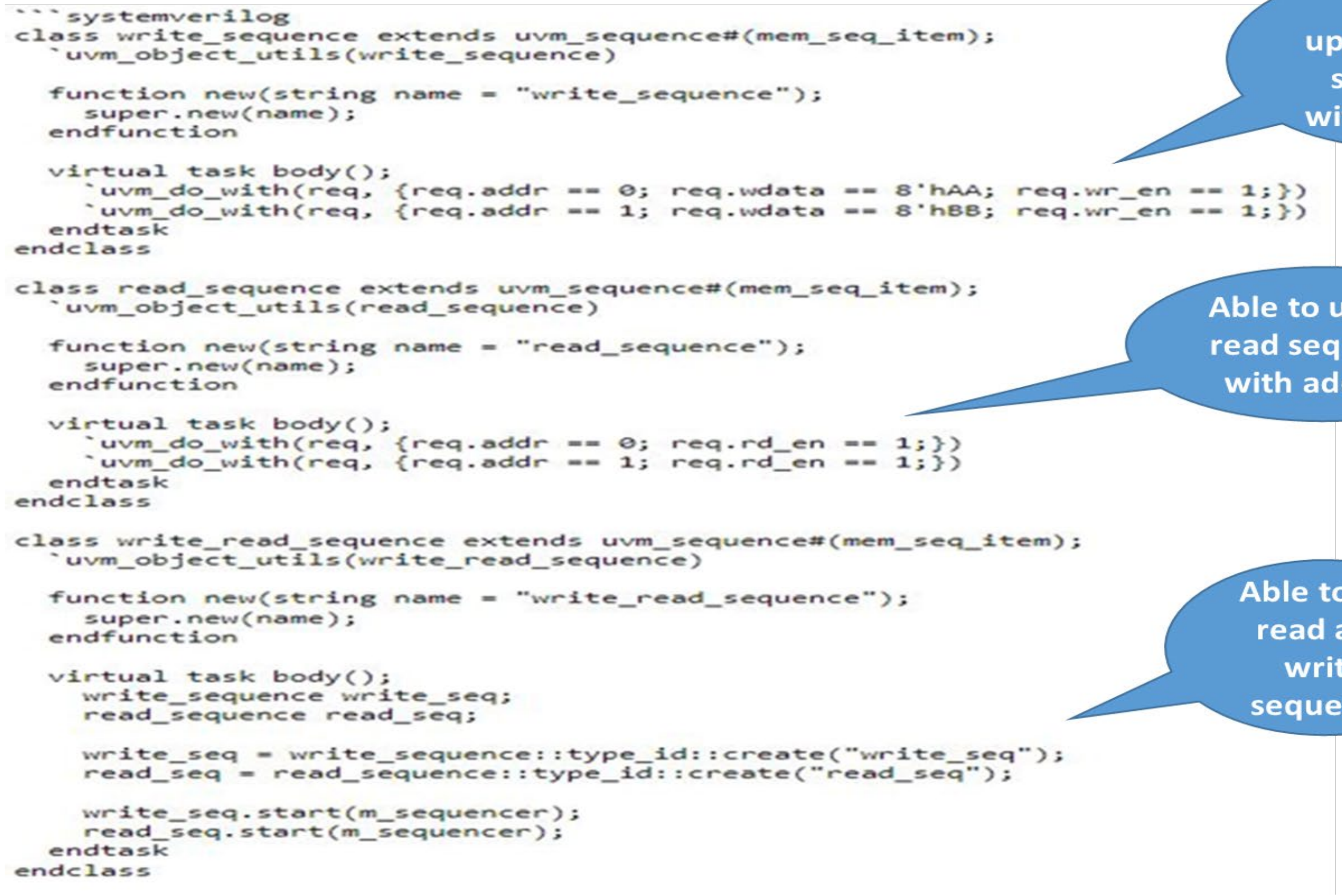
Instruction



```
text1 = "1)The following is sequence item,class mem_seq_item extends uvm_sequence_item;\n//----------------------------------------\n//data and
control fields\n//----------------------------------------\n rand bit [1:0] addr;\nrand bit       wr_en;\nrand bit       rd_en;\nrand bit [7:0]
wdata;\nbit [7:0] rdata;\n//----------------------------------------\n//Utility and Field macros\n//----------------------------------------
\n`uvm_object_utils_begin(mem_seq_item)\n`uvm_field_int(addr,UVM_ALL_ON)\n`uvm_field_int(wr_en,UVM_ALL_ON)\n`uvm_field_int(rd_en,UVM_ALL_ON)\n
`uvm_field_int(wdata,UVM_ALL_ON)\n`uvm_object_utils_end\n//---------------------------------------\n//Constructor\n//-------------------------
--------------\nfunction new(string name = 'mem_seq_item');\nsuper.new(name);\nendfunction\//----------------------------------------
\n//constaint, to generate any one among write and read\n//----------------------------------------\nconstraint wr_rd_c { wr_en != rd_en; };
\nendclass . 2)The following is write read sequence,class write_read_sequence extends uvm_sequence#
(mem_seq_item);\n\n`uvm_object_utils(write_read_sequence)\n//---------------------------------------- \n//Constructor\n//----------------------
-----------------\nfunction new(string name = 'write_read_sequence');\nsuper.new(name);\nendfunction\nvirtual task body();\n`uvm_do_with(req,
{req.wr_en==1;})\n`uvm_do_with(req,{req.rd_en==1;})\nendtask \nendclass. 3)The following is write read test\n, class mem_wr_rd_test extends
mem_model_base_test;\n `uvm_component_utils(mem_wr_rd_test)\n//----------------------------------------\n// sequence instance \n//------------
--------------------------- \n write_read_sequence seq;\n//----------------------------------------\n// constructor\n//-------------------------
--------------\nfunction new(string name = 'mem_wr_rd_test',uvm_component parent=null);\n super.new(name,parent);\n endfunction : new\n//-----
-----------------------------------\n// build_phase\n//----------------------------------------\nvirtual function void build_phase(uvm_phase
phase);\nsuper.build_phase(phase);\n// Create the sequence\n seq = write_read_sequence::type_id::create('seq');\n endfunction :
build_phase\n//----------------------------------------\n// run_phase - starting the test\n//----------------------------------------\n task
run_phase(uvm_phase phase);\nphase.raise_objection(this);\nseq.start(env.mem_agnt.sequencer);\n phase.drop_objection(this);\n//set a drain-
time for the environment if desired\nphase.phase_done.set_drain_time(this, 50);\nendtask : run_phase\n endclass : mem_wr_rd_test"

text=text1+"\n You are an expert system verilog programmer who can understand the above sequence ,test completely.Give me the sequence and
test code to trigger following scenario: write to address 0 followed by write to address 1 followed by reads for the same addresses ,don't
explain anything just give the output."
```

Prompt

Execution and Results : Example 2 (2/2)



~~~
```systemverilog
class write_sequence extends uvm_sequence#(mem_seq_item);
  `uvm_object_utils(write_sequence)

  function new(string name = "write_sequence");
    super.new(name);
  endfunction

  virtual task body();
    `uvm_do_with(req, {req.addr == 0; req.wdata == 8'hAA; req.wr_en == 1;})
    `uvm_do_with(req, {req.addr == 1; req.wdata == 8'hBB; req.wr_en == 1;})
  endtask
endclass

class read_sequence extends uvm_sequence#(mem_seq_item);
  `uvm_object_utils(read_sequence)

  function new(string name = "read_sequence");
    super.new(name);
  endfunction

  virtual task body();
    `uvm_do_with(req, {req.addr == 0; req.rd_en == 1;})
    `uvm_do_with(req, {req.addr == 1; req.rd_en == 1;})
  endtask
endclass

class write_read_sequence extends uvm_sequence#(mem_seq_item);
  `uvm_object_utils(write_read_sequence)

  function new(string name = "write_read_sequence");
    super.new(name);
  endfunction

  virtual task body();
    write_sequence write_seq;
    read_sequence read_seq;

    write_seq = write_sequence::type_id::create("write_seq");
    read_seq = read_sequence::type_id::create("read_seq");

    write_seq.start(m_sequencer);
    read_seq.start(m_sequencer);
  endtask
endclass
~~~

Able to update write sequence with address

Able to update read sequence with address

Able to call read and write sequence

Conclusion

- Deepseek is trained on multiple programming languages as well as UVM framework
- With few shot learning and various prompt inputs, it is possible to generate the output. So, instruction fine-tuning may not be required (unlike other ML tools).
- The model shall help verification engineers to automate sequence, thus making their work easier and reducing the amount of time required to create individual sequences.
- Proposed model can also be extended to generate an ***optimized*** regression testlist and directed testcases for achieving coverage goals

Future work:

- Research to make the model analyze the coverage report and auto-generate the test sequences to achieve the coverage goals

THANK YOU !