Semiconductor Simulation Code Generation

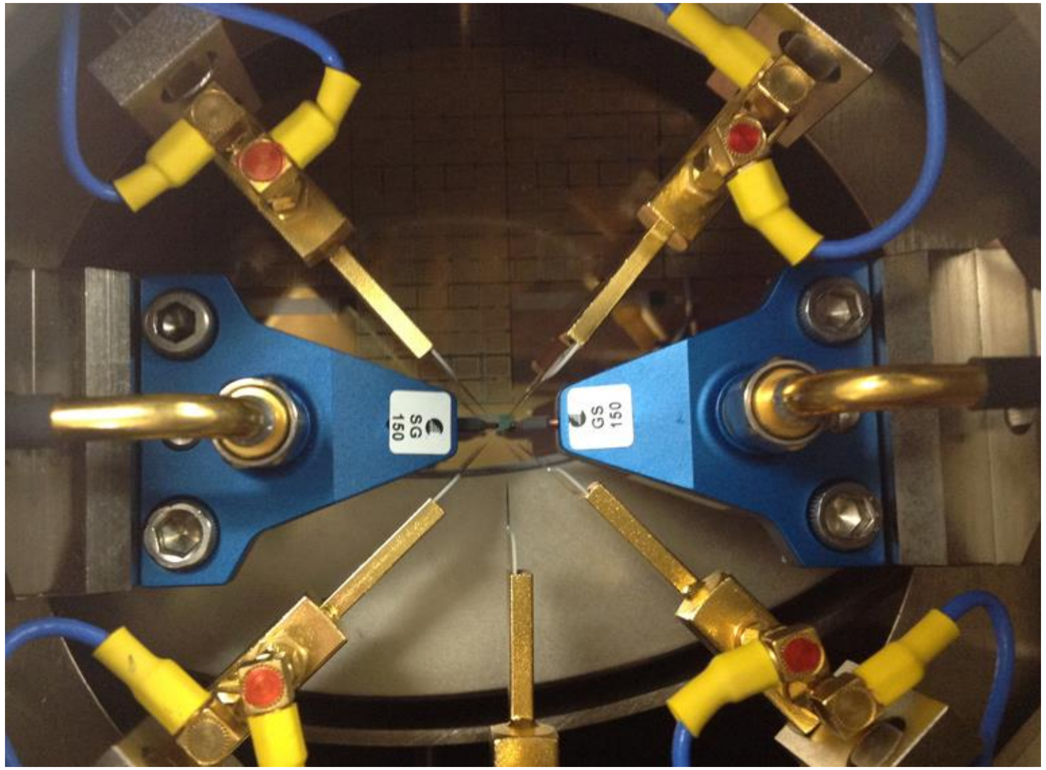
Prompt Engineering Experiments
Applied knowledge from lectures 10-11
Course CSC 575 - Dr. Lin
Zoe Rochelle
12/03/2025

Project Goal

- **Goals**
  - 1) Build a language model system to generate code from natural language specifications
  - 2) then design a benchmark to evaluate model performance.
    - Why does this matter? Faster iteration, fewer syntax/parameter mistakes, template automation
- **Definition of Success:** higher pass-rate on syntax + better semantic correctness on a custom benchmark

What is TCAD, Silvaco, and SPICE-compatible?

- **Technology CAD (TCAD)**: is a software suite that is used to simulate how semiconductor devices are built and behave using physics-based models.
- **Silvaco**: is a company that makes TCAD products
- **SPICE-compatible**: is code that is in the "SPICE netlist" format that circuit simulators can run.



Constraints & Deliverables

- **Model constraint:** no more than **1 Billion parameters**
- **Deliverables:**
  - trained model & tokenizer
  - custom benchmark (at least 20 cases)
  - evaluation scripts
  - report (no more than 4 pages)
  - and a Readme – (I think)

Data I'll Be Using

silvaco_dataset_train.json (instruction-code pairs)

The **.in** and **.lib** files

2. `Silvaco_Examples_Student.zip` (18MB)

Reference examples from Silvaco user manuals (optional resource).

Contents:

- 726 `.in` files: Silvaco input deck files
- 76 `.lib` files: Model library files
- PDF manuals: Reference documentation

Purpose:

- Additional context and examples for understanding Silvaco syntax and circuit patterns
- **You can use these .in files to create your own training data** by writing corresponding instruction descriptions
- Provides opportunity for data augmentation and creative exploration

Methodology Overview

Pipeline diagram

I will start with a set of natural-language specifications: like the device or circuit type.

Then apply a prompt template – which is a re-usable, structured input with parameters, I.e.

```
your task {}, constraints:{}, output format: {}
```

Then run the model to generate the TCAD code.

After generation, I will run a series of benchmarking checks: syntax, completeness, correctness – to compute a score.

1. Input specifications (NL)
2. Prompt template
3. LLM generate code
4. Post-checks (like syntax)
5. Benchmark vs score

Prompt Eng Experiments - What I’ll Be Comparing

1. **Baseline instruction prompt** (the control)
2. **Provide structured output and add hard constraints**
3. **Few-shot prompting** (2–3 curated examples)
4. **Use the "Critique, then revise" technique**
   - Pass 1: generate code
   - Pass 2: run a checklist inside the prompt to revise the output

Benchmark Design & Metrics

Benchmark requirements:

- At least 20 custom test cases
- Use diverse (TCAD) device types, complexity, and edge cases
- Use at least 3 metrics with justification

Metrics I will be measuring:

- Syntax validity pass rate
- Parameter exact-match for key fields (like regions, doping, electrodes, models)
- Build a 'component and section coverage' score (ask: 'does the output include the required parts for a valid TCAD input?')

Tools I'll be using

- **Python + Jupyter** for experiment orchestration
- **Hugging Face Transformers** for model loading, generation (and fine-tuning if I have the time)
- **Evaluation harness** for custom scripts to run prompts across benchmark and score metrics
- **RAG** (as suggested in the project outline) for embeddings + vector store
- Maybe **LoRA/QLoRA**, but only if I have the time to figure out how to use it
- For the model - **Qwen3** (because it was recommended in the outline)

Preliminary Results

I haven't begun running anything so…

I hypothesize that

- the **baseline test** will not perform very well
- **Few-shot prompting** will improves syntax and completeness
- That the **critique/revise** phase will improve the coverage and semantic alignment

Timeline / Next Steps / TODOs

- Work out the baseline prompt and build benchmark JSON
  - using 20+ cases
- Implement metrics and run the prompt variants
- Iterate then determine the highest performing prompt and run a failure analysis
- Submit the code and the report

End