Student Project Ideas from Modelon – Spring 2026

Project A — Baby LLM for Modelica (Fully Local, Fully Open)

Objective: Train or fine-tune and evaluate a <150M LLM to generate simple Modelica code from natural-language instructions; run entirely on browser WebGPU or laptop CPU/GPU.

Core Tasks:

- Benchmark small pretrained LLMs: TinyLlama-110M, GPT-2 Small, MobileLLM-60M.
- Prepare Modelica instruction data from MSL and public repos; build prompt→code pairs.
- Instruction fine-tune with LoRA (PEFT); optionally quantize with BitsAndBytes.
- Evaluate syntax correctness and template generation on basic models.
- Deploy locally via WebLLM/MLC-LLM or llama.cpp (GGUF).

Practical References:

- TinyLlama https://github.com/jzhang38/TinyLlama
- GPT-2 Small https://huggingface.co/gpt2
- MobileLLM-60M https://arxiv.org/abs/2402.14905
- PEFT (LoRA) https://github.com/huggingface/peft
- WebLLM / MLC-LLM https://github.com/mlc-ai/web-llm
- Modelica Standard Library https://github.com/modelica/ModelicaStandardLibrary

Deliverables:

- Fine-tuned mini-LLM and training scripts.
- Local inference demo (browser or CLI).
- 6–8 page technical report.

Success Criteria:

- Train or fine-tune a compact LLM on Modelica-related prompt→code examples.
- Generate a new component model, e.g.:
  - *Modelica.Mechanics.Rotational*: simple gear, clutch, spring/damper element
  - *Modelica.Fluid*: simple valve, pipe segment, or boundary model
- Create a new system model using the generated component.
- Run a successful simulation in Modelon Impact (access to impact.modelon.cloud provided).
- Demonstrate the model running inference locally (browser or CLI).

Project B — Datasheet → Structured Parameters Extraction

Objective: Build a lightweight PDF-processing pipeline that extracts engineering parameters (tables, key-value entries) and outputs normalized JSON suitable for simulation input. Data sheets and simulation models will be provided.

Core Tasks:

- Detect tables/text blocks using LayoutParser with DocLayNet pretrained models.
- Extract tables with Camelot (or PubTables-1M structures).
- Extract key-value specs via Donut (OCR-free) or Tesseract+regex.
- Normalize units (bar, MPa, mm) and emit JSON.

Practical References:

- LayoutParser https://layout-parser.github.io
- DocLayNet (models) https://huggingface.co/datasets/impactvisionlab/doclaynet
- Camelot https://camelot-py.readthedocs.io
- PubTables-1M https://arxiv.org/abs/2110.00061
- Donut (OCR-free) https://huggingface.co/naver-clova-ix/donut-base
- Tesseract OCR https://github.com/tesseract-ocr/tesseract

Deliverables:

- PDF→JSON extraction pipeline + visual overlays.
- Small evaluation on 5–10 datasheets.
- 5–7 page report.

Success Criteria:

- A working tool that detects tables, key-values, and units from provided datasheets.
- Normalized JSON output of parameters ready for ingestion into Modelica models.
- Demonstration that the resulting JSON can be successfully read into an existing Modelon model, e.g., a pump, motor, or valve.
- Verification using the provided data sheets.

Project C — Graph Digitization (Image → XY Data)

Objective: Implement a computer-vision workflow that converts plotted performance curves into numerical (x,y) data suitable for simulation and analysis. Plotted performance curves will be provided.

Core Tasks:

- Locate graph region via thresholding/contours (OpenCV).
- OCR axis labels/ticks with Tesseract to derive scaling.
- Segment curve using Canny + contour filtering; handle colored curves.
- Map pixel coordinates → real values; export CSV/JSON; overlay for QA.

Practical References:

- OpenCV https://opencv.org
- Tesseract OCR https://github.com/tesseract-ocr/tesseract
- ChartOCR (reference repo) https://github.com/SoothsayerTechnology/ChartOCR

Deliverables:

- XY extraction tool with visual comparisons.
- Tested on at least 5 graphs from real datasheets.
- 5–7 page report.

Success criteria:

- Detection and extraction of graph regions, axes, and tick labels.
- Extraction of (x,y) data points for at least one curve (e.g., efficiency vs. speed).
- Automatic mapping from pixel coordinates to real physical units.
- Export of numeric results as CSV or JSON.
- Optional but recommended: use the extracted curve in a simple simulation, e.g., interpolating a lookup table in a machine or drive model.