**Revised Hackathon MVP Specification: AI-Assisted Hardware-Aware Model Profiler**

Yes, this spec is very close to the prototype I proposed for "AccelForge" in our previous discussion—it's essentially an expanded, detailed version of the same core idea. The focus on uploading an ONNX model, parsing it into descriptors, using MoE for performance prediction, a lightweight NAS-inspired search for configs, and visualizing results in a Streamlit app aligns perfectly with automating hardware-aware optimization without full synthesis. It stays "tech-first" by emphasizing AI-driven hardware R&D (e.g., predicting accelerator configs), drawing from frontiers like NAS and MoE, and avoiding humanitarian angles.

To make it achievable in 18 working hours (assuming a 2-4 person team with ML/hardware basics), I've tailored it by:

* **Simplifying Scope**: Reduce complexity in MoE (use pre-built or simple regressors instead of full training), limit search space (fewer config params, 10-20 iterations max), and use proxies for simulation (e.g., FLOPs-based estimates vs. cycle-accurate).
* **Prioritizing Core Features**: Ensure the MVP loop (upload → parse → predict/search → visualize) works end-to-end with 2-3 demo models. Cut non-essentials like full RTL export to stretch goals.
* **Jury Appeal**: Emphasize innovation (NAS+MoE combo as "AI-for-AI-hardware"), demo-ability (quick runs on laptop, interactive sliders), and impact (e.g., "Cuts exploration time from days to minutes"). Add polish like error handling and benchmarks to show reliability. Judges from NVIDIA/Qualcomm/Intel love measurable tech depth (e.g., "10% error on predictions") over fluff.
* **Time Buffer**: Extended from 12-16h to 18h by adding testing/polish hours, but kept realistic—no overambition.

The revised spec stays true to principles: Lightweight "what-if" simulator for ML deployment on hardware, using AI (MoE/NAS) for predictions/recommendations, focused on efficiency without heavy tools.

**1. Problem Context**

Deploying ML models efficiently on hardware is challenging due to varying model architectures (layers, tensor sizes, FLOPs) and hardware specs (compute cores, memory bandwidth, power). Users need to balance latency, memory, and power under constraints (e.g., "<1GB VRAM, <20ms inference"). Tools like hls4ml, FINN, Vitis AI, or TVM are powerful but require full compilation/synthesis, which is too slow/heavy for quick prototyping or hackathons. Our MVP provides a fast, AI-driven simulator and recommender to predict hardware-model mappings, enabling rapid exploration.

**2. High-Level Solution**

A Streamlit web app where users upload a pretrained ML model (ONNX format). The backend:

* Parses the model into structural descriptors (layers, FLOPs, params).
* Generates a JSON workload summary.
* Simulates hardware configs (e.g., batch size, precision) using a simple MoE for performance predictions.
* Runs a lightweight search (NAS-lite) to find optimal configs under constraints.
* Outputs: Tables, Pareto plots, and a top recommendation.

This acts as a "what-if" tool for ML engineers, predicting deployment feasibility without real hardware or synthesis.

**3. Workflow (Step by Step)**

**Step A: Input**

* User uploads ONNX model (support 2-3 common ones like MobileNet, ResNet-18 for demo; handle others gracefully).
* Constraints via sliders: Max latency (ms), max memory (MB), max power (W) (use defaults if not set).

**Step B: Parsing → JSON Representation**

* Use ONNX Runtime to load and parse the graph.
* For each layer: Extract type (e.g., Conv2D, GEMM), input/output shapes, param count.
* Compute FLOPs per layer (simplified formulas: e.g., Conv2D FLOPs = 2 \* Cin \* Cout \* K^2 \* Hout \* Wout).
* Output JSON:

text

{

"layers": [

{"type": "Conv2D", "in\_shape": [3,224,224], "out\_shape": [64,112,112], "params": 9408, "flops": 118000000},

{"type": "ReLU", "in\_shape": [64,112,112], "out\_shape": [64,112,112], "params": 0, "flops": 800000}

],

"totals": {"params": 11700000, "flops": 1800000000}

}

* (Simplified: Skip complex ops like attention for MVP; fallback to total FLOPs if parsing fails.)

**Step C: Workload Mapping**

* Map JSON to abstract workloads: e.g., total MACs (from FLOPs), memory accesses (tensor sizes \* reads/writes proxy, e.g., 2 \* sum(tensor volumes)).
* Define hardware configs as params: Batch size (1,4,8,16), precision (fp32, fp16, int8), simulated array size (e.g., 4x4, 8x8 tiles as compute proxy).

**Step D: AI Prediction (MoE)**

* Input: Workload JSON + config (e.g., batch=8, precision=fp16).
* Use a simple MoE: Gating network (basic MLP) routes to 3 experts (scikit-learn regressors for latency, memory, power).
* Predictions based on proxies:
  + Latency ~ FLOPs / (array\_size \* clock\_proxy \* batch) + overhead.
  + Memory ~ params \* precision\_bytes + tensor\_buffers.
  + Power ~ latency \* power\_per\_flop\_proxy (use synthetic coeffs from benchmarks).
* (Simplified: "Train" experts on 50-100 synthetic data points generated in-code, e.g., vary FLOPs/params and measure proxies. No full PyTorch training loop—use fit() on arrays.)

**Step E: NAS-lite Search**

* Search space: 10-20 configs (grid over batch, precision, array size).
* For each: Run MoE to predict metrics.
* Filter feasible ones (meet constraints).
* Rank Pareto-optimal (e.g., minimize latency under memory/power caps using simple sorting).

**Step F: Output**

* Table: Configs with predicted latency/memory/power.
* Plot: Scatter (latency vs. memory) with Pareto frontier highlighted (Plotly).
* Recommendation: Top config (e.g., "Batch=8, FP16, 8x8 array: 15ms latency, 800MB memory").
* (No full RTL; stretch for basic JSON mapping like {"op": "Conv2D", "tile": "8x8"}.)

**4. MVP Timeline (Hackathon 18h)**

Adjusted for more testing/polish to ensure features work reliably (jury loves demos without crashes).

| **Phase** | **Tasks** | **Estimated Time** | **Tools/Libs** |
| --- | --- | --- | --- |
| **Setup & Parsing (Hours 1-3)** | - Env setup (Python, libs). - Write ONNX parser + FLOPs calculator. - Test with 2-3 sample models (download from ONNX Zoo). | 3 hours | ONNX Runtime, NumPy |
| **Workload + MoE Prediction (Hours 4-9)** | - Map to workloads. - Build simple MoE: Generate synthetic data (e.g., random FLOPs/params → proxy metrics). - Fit regressors; implement gating/routing. | 6 hours | Scikit-learn (regressors), PyTorch (if needed for gating MLP) |
| **Search + UI Build (Hours 10-13)** | - Implement grid search + filtering/ranking. - Streamlit: Upload, sliders, button, display table/plot. | 4 hours | Streamlit, Plotly |
| **Testing & Debug (Hours 14-16)** | - End-to-end tests: 3 models, vary constraints. - Error handling (e.g., invalid ONNX → message). - Benchmark accuracy (compare predictions to real ONNX Runtime inference on CPU/GPU). | 3 hours | Manual runs, logging |
| **Polish & Pitch Prep (Hours 17-18)** | - Add UI flair (tooltips, loading spinner). - Slides: Problem, demo video, metrics (e.g., "Predictions within 15% of real"). - Script 2-min demo. | 2 hours | Google Slides, screen record |

**Total Effort**: 18 hours. If behind, drop power prediction (focus on latency/memory) or fix search to 10 configs.

**5. Success Metrics**

* **Technical**: Parses ONNX accurately; predictions within 15% error vs. real ONNX Runtime measurements (demo with benchmarks).
* **Innovation**: NAS-lite + MoE as a novel "AI optimizer for accelerators"—highlight how it scales to real tools like TVM.
* **Impact**: "Accelerates hardware exploration by 10x for edge AI devs." Jury appeal: Quantifiable (e.g., "Search in <10s"), extensible (to real NPUs), and demo-friendly (interactive app runs smoothly on laptop).
* **Reliability**: All features work—tested paths, no crashes on demo models.

**6. Tools / Stack**

* **Backend**: Python, ONNX Runtime (parsing/inference), scikit-learn (MoE experts), NumPy (computations).
* **Frontend**: Streamlit (UI, sliders, tables/plots via Plotly).
* **Data**: Synthetic benchmarks (generate in-code; optional: Quick CPU measurements for 10 data points).
* **Docs**: Google Slides for pitch.

**7. Stretch Goals**

* Basic RTL skeleton JSON (e.g., map layers to tiles).
* Sparse MoE toggle (for int8 configs).
* Support more models (e.g., BERT variants).

**🔑 One-liner Vision**

A hackathon MVP that uploads an ML model, predicts its hardware performance under constraints, and recommends optimized configs—bridging AI research with accelerator design without synthesis overhead.