Yes — but do NOT build your own physics engine from scratch. That would be a multi-year, multi-million-dollar effort (think the scale of Ansys, dSPACE, or IPG CarMaker teams). Automotive physics simulation is an extremely mature field with battle-tested open-source engines that already outperform what a small team could build in reasonable time/budget.

Instead, **extend an existing SOTA open-source physics simulator** — this is exactly how top labs and companies (Waymo, Tesla, Uber ATG, etc.) generate synthetic diagnostics data today.

Best Options for Your Use Case (2026)

Simulator	Why It's Perfect for Diagnostics	Fault Injection Ease	Open-Source?	Fidelity Level	Key Links
Project Chrono (top recommendation)	Multi-body vehicle dynamics + deformable terrain + sensors + fluids + FEA. Excellent for engine/transmission/wheel faults, OBD-like signals. Used by NIRA Dynamics for real diagnostics pipelines.	Very easy (template-based + Python/C++ API)	Fully open (Apache 2.0)	Very high (validated against real vehicles)	projectchronc Chrono::Vehic
CARLA	Urban driving + perfect sensor sim (LiDAR, camera, IMU, CAN/OBD export) + already used for massive synthetic datasets.	High — community fault plugins exist	MIT	High for perception + dynamics	carla.org
BeamNG.tech	Hyper-realistic soft-body vehicle deformation + crashes + tire physics. Great for structural/ suspension faults.	Excellent (Lua scripting for faults)	Academic license (free for research)	Highest for vehicle damage	beamng.tech
Open-Car-Dynamics (TUM)	Lightweight multibody ODE-based vehicle model. Perfect for quick trajectory + controller testing.	Easy	Apache 2.0	Medium-high	github.com/TI Car-Dynamics

How to Generate World-Class Synthetic Diagnostics Data (Step-by-Step)

1. **Choose base simulator**
   Start with **Project Chrono** (best balance of fidelity + extensibility for engine/OBD faults).

2. **Add fault injection** (this is the key part for diagnostics)

```python
# Example using Chrono Python API (or C++)
import pychrono as chrono
import pychrono.vehicle as veh

# Create vehicle
vehicle = veh.WheeledVehicle(...)

# Inject faults
def inject_fault(t):
    if t > 100:  # e.g., after 100s
        # Sensor drift (O2 sensor)
        vehicle.GetSensor("O2").AddNoise(0.15)  # 15% bias
        # Actuator failure (throttle stuck)
        vehicle.GetActuator("throttle").SetStuck(0.8)
        # Engine misfire
        vehicle.GetEngine().SetMisfireRate(0.3)

# Run simulation loop with fault callback
```

   Common faults to simulate:

   - Sensor: drift, noise, stuck, dropout (MAF, O2, coolant temp, etc.)
   - Actuator: stuck, delayed, partial failure
   - Mechanical: tire puncture, suspension wear, bearing failure
   - Electrical: short/open circuit, ECU glitch

3. **Export OBD-II style data**
   Log exactly what real OBD tools see: PID values (RPM, speed, temps, pressures, DTCs, freeze-frame). Chrono/CARLA can output CAN/OBD-compatible logs.

4. **Scale to millions of examples**
    - Domain randomization: vary vehicle models, weather, road conditions, manufacturing tolerances.
    - Run on GPU cluster → generate 10M+ labeled traces in days.
    - Use teacher models (o1-pro / Claude-3.7) to generate diagnostic reasoning traces on top (like OpenThoughts3 style).
5. **Bridge sim-to-real gap** (critical for production)
    - Add realistic sensor noise models (from real OBD datasets you already have).
    - Fine-tune final ML model on 10% real + 90% synthetic (standard sim-to-real recipe).
    - Physics-informed ML: embed Chrono equations directly into your neural net (via NeuralPDE or DiffSim).

Proof This Works (Recent Research)

- NIRA Dynamics (real automotive company) replaced their Simulink models with Chrono for diagnostics testing — huge maintenance win.
- Papers show physics-sim + ML beats real-data-only on fault detection (e.g., SEAD method, hybrid physics-ML ensembles).
- CARLA/BeamNG used successfully for OBD-style anomaly detection datasets.

Bottom line: Building your own physics engine = 5+ years and $10M+. Extending Chrono/CARLA = 1–3 months to first 1M labeled examples, then iterate.

If you want, I can give you a full starter repo template (Chrono + fault injection + data export pipeline) or help pick the exact faults to simulate first based on your target vehicles. What's your priority — engine faults, EV battery, or full vehicle?