| **Metric** | **Traditional EKF** | **Full‑Batch KD dEKF** | **Selective KD dEKF** |
| --- | --- | --- | --- |
| **MSE after training** | 0.100 | 0.050 | ~0.045 |
| **Knowledge Absorbed** | 0 % | 56 % | ~60 % |
| **Risk of Over‑tuning / Drift** | n/a | Medium | Low |
| **Susceptibility to Plateau** | High | High once tuned | Low—each batch on fresh outliers |

Here are concise problem‑statement formulations for the **DSP** and **Model‑Based Design** aspects of your project:

**1. DSP Problem Statement**

**“Design and implement a real‑time digital signal processing front‑end on an ESP32 that conditions 6‑DoF IMU data for downstream knowledge‑distillation filtering. Specifically:**

* **a. Noise Filtering:** Develop IIR/FIR filters (e.g. low‑pass and notch) to remove sensor noise and mechanical vibration from 6‑axis accel/gyro at 200 Hz.
* **b. Feature Extraction & Event Detection:** Compute real‑time statistics (e.g. variance, spectral energy) and implement an “outlier” detector that flags significant motion events (roll > θ, jerk spikes) for selective KD triggers.
* **c. Fixed‑Point Optimization:** Port all DSP routines to CMSIS‑DSP using Q15/Q31 arithmetic to meet a worst‑case compute budget of < 1 ms per sample under FreeRTOS.”

**2. Model‑Based Design (MBD) Problem Statement**

**“Use a model‑based design workflow to specify, simulate, and auto‑generate embedded C code for the differentiable EKF (dEKF) L1 filter on the ESP32, ensuring bit‑exact behavior between simulation and target hardware. Specifically:**

* **a. System Modeling:** Build a Simulink (or equivalent) model that encapsulates the IMU kinematics (f), measurement model (h), and trainable covariance updates (Q, R) in block form.
* **b. Verification & Validation:** Perform MIL/SIL/PIL testing to verify that the simulated dEKF output matches the generated code running on the ESP32 within ±1 % numerical error, and that real‑time deadlines (200 Hz) are met.
* **c. Automatic Code Generation:** Configure code‑generation settings to produce optimized, maintainable C for FreeRTOS—complete with generated parameter‑tuning hooks for micro‑SGD-based covariance updates.”

These two problem statements crisply capture the **DSP** and **MBD** challenges in your project scope, and directly tie into your multi‑path KD architecture.

***Hybrid : Differentiable + Adaptive***

**In the sequential architecture; there are four approaches: -**

**1) Adaptive First: The raw 6-DoF IMU data streams thru the adaptive filter (AF) first, then the differentiable filter (DF). So, the distillation loss calculated between L1 and L2 accounts for AF inference. The LMS algo runs agnostic of DF performance.**

**2) Differentiable First: The raw 6-DoF IMU data streams thru the DF first, then the AF. So, the LMS algo on the AF accounts for DF performance. The KD on DF runs agnostic of AF performance.**

**3) Bistate Inference: Periodically, or smartly, sequence of the filters is switched to optimize the 10 L1 Evaluation Criteria (10L1EC, henceforth).**

**4) Serial Inference Parallel Learning (SIPL): the final filtering is passed thru both, but the individual learning of both DF and AF is mutually exclusive.**

**love this. here’s a clean, paper-ready way to define and compute your KD efficiency—and tie it to temporal (runtime/energy) and spatial (memory) complexity.**

**Metric suite**

**1) Knowledge Absorption (KA)**

**Quantifies how much of the teacher’s task performance the student preserves.**

**Pick one or more task metrics mm that matter for IMU denoising / downstream estimation:**

* **Denoising: MAE, RMSE, SNR gain, PSD mismatch (e.g., L2L\_2 between spectra), Allan-deviation fit error.**
* **Estimation: attitude/heading error (deg), drift rate (deg/hr), APE/RPE for trajectories, bias estimation error.**

**For a metric where lower is better (e.g., RMSE), define improvement vs a noisy baseline BB:**

**ImpX=m(B)−m(X)m(B)×100%,X∈{Teacher (T),Student (S)}\text{Imp}\_X = \frac{m(B) - m(X)}{m(B)}\times 100\%, \quad X\in\{\text{Teacher (T)},\text{Student (S)}\}**

**Then**

**%KAm=ImpSImpT×100%\%KA\_m = \frac{\text{Imp}\_S}{\text{Imp}\_T}\times 100\%**

**For metrics where higher is better (e.g., SNR), swap the sign:**

**ImpX=m(X)−m(B)m(B)×100%\text{Imp}\_X = \frac{m(X) - m(B)}{m(B)}\times 100\%**

**Aggregate across metrics (weights wmw\_m sum to 1):**

**%KA=∑mwm⋅%KAm\%KA = \sum\_m w\_m \cdot \%KA\_m**

**Tip: set wmw\_m based on your claims (e.g., 0.6 for attitude error, 0.4 for spectral fidelity).**

**2) Model Compression (MC)**

**Quantifies how much smaller/faster the student is. Compute separately for space and time:**

* **Spatial compression (memory/size):**

**%MCmem=(1−MemSMemT)×100%\%MC\_{\text{mem}}=\left(1-\frac{\text{Mem}\_S}{\text{Mem}\_T}\right)\times 100\%**

**where Mem = model bytes on disk or peak RAM at inference.**

* **Temporal compression (compute cost):**

**%MCtime=(1−CostSCostT)×100%\%MC\_{\text{time}}=\left(1-\frac{\text{Cost}\_S}{\text{Cost}\_T}\right)\times 100\%**

**where Cost can be FLOPs per second of signal, wall-clock latency, or energy per second (mJ/s).  
If you can, report both latency and energy; else at least FLOPs or latency on a fixed device.**

**You can also define a combined compression (weighted):**

**%MC=α⋅%MCtime+(1−α)⋅%MCmem\%MC = \alpha\cdot \%MC\_{\text{time}} + (1-\alpha)\cdot \%MC\_{\text{mem}}**

**with α∈[0,1]\alpha\in[0,1] (e.g., α=0.6\alpha=0.6 for edge-first scenarios).**

**3) Core efficiency ratio**

**Your original idea:**

**E=%KA%MCE = \frac{\%KA}{\%MC}**

**Use the combined %MC\%MC above (or report two ratios Etime,EmemE\_{\text{time}}, E\_{\text{mem}}).**

**Interpretation:**

* **E>1E>1: student gains more knowledge (relative) than the compression it suffered → efficient.**
* **E≈1E\approx1: balanced trade-off.**
* **E<1E<1: over-compressed; lost too much knowledge.**

**Complexity mapping (make it apples-to-apples)**

**Student (LMS + shallow ANN)**

* **LMS (per sample): O(L)O(L) MACs, where LL = FIR length (+ O(L)O(L) for coefficient update if adaptive every sample).**
* **ANN (per sample): sum of layer MACs. For 1D conv with Cin→CoutC\_{in}\to C\_{out}, kernel kk, receptive field rr, FLOPs ≈2⋅k⋅Cin⋅Cout\approx 2\cdot k\cdot C\_{in}\cdot C\_{out} per position; for FC layers 2⋅nin⋅nout2\cdot n\_{in}\cdot n\_{out}.**
* **Memory: LMS weights LL + ANN params (few 10–100k typical for “shallow”).**

**Classical filters (for context in your paper)**

* **KF: per step O(n3)O(n^3) due to covariance update/inversion (with state dim nn).**
* **EKF/UKF: similar cubic components + Jacobians/sigma points.**
* **PF: O(Np)O(N\_p) per step with NpN\_p particles (often hundreds–thousands).  
  These make a clear edge-efficiency case vs your shallow ANN.**

**Experimental protocol (so results are defensible)**

1. **Datasets: use train/test splits; if no GT trajectories, evaluate denoising + downstream attitude with a reference (e.g., VIO) or synthetic GT injection.**
2. **Teacher: train mbrossar’s model (or another high-capacity network) on train set; fix weights.**
3. **Student: LMS+ANN distilled on same train split using soft targets (teacher outputs).**
   * **Loss: L=λ1∥yS−yT∥22+λ2\mathcal{L}=\lambda\_1\|y\_S-y\_T\|\_2^2 + \lambda\_2 (distributional/PSD loss, optional).**
   * **Freeze/test on the held-out test set.**
4. **Baselines: raw, LMS-only, KF/EKF/UKF/PF (if applicable), ANN-only.**
5. **Measure:**
   * **Task metrics mm on test set → compute %KA\%KA.**
   * **Mem: serialized model size and peak inference RAM.**
   * **Time/Energy: average per-second signal latency and (if possible) device power to get mJ/s.**
   * **Compute %MCtime,%MCmem,%MC\%MC\_{\text{time}}, \%MC\_{\text{mem}}, \%MC, then EE.**

**Reporting**

**Primary table**

* **Columns: Method, Task metric(s), %KA\%KA, Model size (MB), Latency (ms/s), Energy (mJ/s), %MCmem\%MC\_{\text{mem}}, %MCtime\%MC\_{\text{time}}, EmemE\_{\text{mem}}, EtimeE\_{\text{time}}, EE.**

**Plots**

* **Pareto curve: %KA\%KA vs %MC\%MC (higher-right is better).**
* **Bar: latency/energy per method.**
* **PSD/Allan overlays of raw vs teacher vs student.**

**Example (numbers just illustrative)**

* **Baseline RMSE = 10.0; Teacher RMSE = 5.0; Student RMSE = 5.6.  
  ImpT=(10−5)/10=50%\text{Imp}\_T=(10-5)/10=50\%, ImpS=(10−5.6)/10=44%\text{Imp}\_S=(10-5.6)/10=44\% → %KA=88%\%KA=88\%.**
* **Teacher size 40 MB, Student size 4 MB → %MCmem=90%\%MC\_{\text{mem}}=90\%.**
* **Teacher latency 8 ms/signal-sec, Student 1.2 ms → %MCtime=85%\%MC\_{\text{time}}=85\%.**
* **α=0.6⇒%MC=0.6⋅85+0.4⋅90=87%\alpha=0.6\Rightarrow \%MC=0.6\cdot85+0.4\cdot90=87\%.**
* **E=%KA/%MC=88/87=1.01E=\%KA/\%MC=88/87=1.01 → near-perfect tradeoff (excellent for edge).**

**Implementation mini-snippets**

**Compute KA (Python)**

**def ka\_fraction(metric\_baseline, metric\_teacher, metric\_student, higher\_is\_better=False):**

**if higher\_is\_better:**

**imp\_t = (metric\_teacher - metric\_baseline) / metric\_baseline**

**imp\_s = (metric\_student - metric\_baseline) / metric\_baseline**

**else:**

**imp\_t = (metric\_baseline - metric\_teacher) / metric\_baseline**

**imp\_s = (metric\_baseline - metric\_student) / metric\_baseline**

**return max(0.0, imp\_s / max(1e-12, imp\_t)) # fraction (0..inf)**

**# aggregate (weights sum to 1)**

**def KA(metrics):**

**return 100.0 \* sum(w \* ka\_fraction(\*\*args) for w, args in metrics)**

**Compression & Efficiency**

**def compression(a, b): # a=student, b=teacher**

**return 100.0 \* (1.0 - a / b)**

**MC\_mem = compression(Mem\_S, Mem\_T)**

**MC\_time = compression(Time\_S, Time\_T) # or Energy\_S/Energy\_T**

**MC = 0.6 \* MC\_time + 0.4 \* MC\_mem**

**E = KA\_value / MC**

**Practical tips for your setup**

* **Keep the ANN shallow (e.g., 2–3 conv1d layers + small FC) and quantify MACs/FLOPs; this will make your %MC\_time shine.**
* **Use fixed-point (int8) quantization post-training for even larger %MCmem\%MC\_{\text{mem}} with minimal KA loss; report both FP32 and INT8.**
* **For fairness, run all methods on the same edge device and pin the thread count; measure latency with warm-ups.**
* **Include ablation: LMS-only, ANN-only, LMS+ANN (no KD), LMS+ANN (KD). KD should dominate.**

**If you want, I can turn this into a small evaluation harness (Python) that ingests your CSVs + teacher/student outputs, computes KA/MC/E, and dumps a ready-to-publish table and plots.**