Stress Detection using Context-Aware Sensor Fusion from Wearable Devices

IEEE Internet of Things Journal, , 2023

Cited by: 53

Presenter: Nooshin Taheri

10/8/2025

Background & Limitations of Existing Stress Detection Methods

- **Wearable health devices** are increasingly used for **stress detection**, relying on multiple physiological sensors (e.g., ECG, EDA, EMG, BVP, TEMP).

- **Stress detection methods** face two major challenges:
  - **Lack of robustness –** sensor measurements are noisy and degrade model performance.
  - **Lack of adaptation –** static model architectures cannot adjust to changing sensing conditions (the *noise context*).

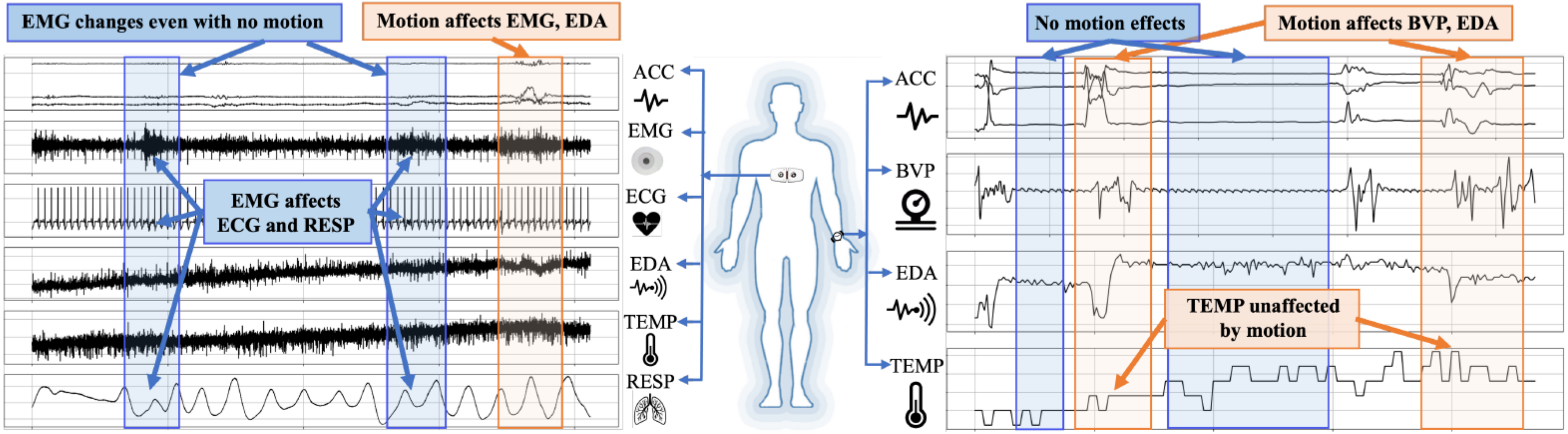
- **Stress classification techniques:**
  - **Deep learning models** can capture **temporal patterns** in sensor data.
  - **Classical machine learning models** are **more commonly used** in stress detection.
  - **Classical models are simpler and less computationally demanding**, making them **better suited for on-device deployment** in wearable systems.

  - **Limited coverage:** Single sensor modalities capture only part of the stress response.
  - **Static fusion:** Combining all sensors without context can worsen accuracy.

- ➡️ **Gap:** A **context-aware, adaptive fusion framework** is needed to dynamically select reliable sensors based on their noise context.

Why Context-Aware Sensor Fusion Is Needed

The context of noise on sensors varies depending on the location of the wearable device.



a) Time segment (60s) for a *Baseline* class from Chest data

b) Time segment (60s) for a *Baseline* class from Wrist data

- **Sensor noise varies by device location → ** Motion affects wrist sensors, while muscle contractions affect chest sensors.
- **Blind fusion can mislead the model**
  → A **context-aware fusion approach** is needed to adapt to changing noise conditions.

Contributions

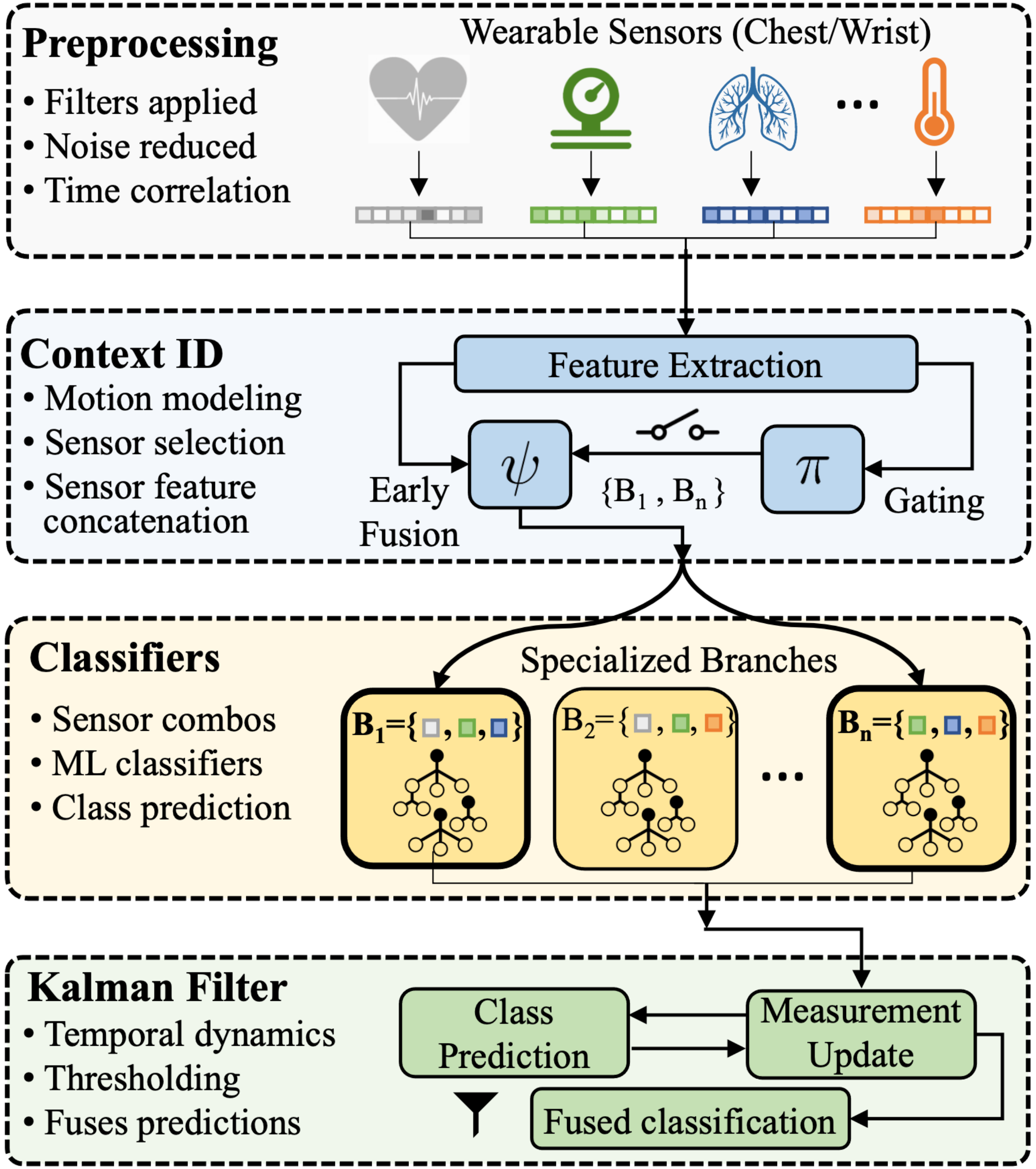
- **Introduced SELF-CARE**
  - a *generalized selective sensor fusion framework* for stress detection from wearable devices.
- **Proposed context identification**
  - models the *noise context* (motion for wrist, muscle contraction for chest) to adaptively select reliable sensors.
- **Developed a novel late-fusion method**
  - uses a **Kalman filter** to incorporate temporal dynamics and improve classification stability.

Problem Formulation – Need for Adaptive Sensor Selection

- The **noise context** varies by device location —
  wrist sensors are affected by **motion**, chest sensors by **muscle contractions**.
- **Fixed (static)** models treat all sensors equally → performance drops when some signals are noisy.
- The system must **adaptively choose which sensors to fuse** depending on the current context.
- SELF-CARE formulates stress detection as a **context-driven adaptive fusion problem**, rather than a static classification task.
- Introduces two key modules:
  - **Gating model (π):** detects the current *noise context* from ACC (wrist) or EMG (chest).
  - **Selection mechanism (ρ):** picks the best subset of models/sensors (**ϕ***).
- Goal: **maximize stress detection accuracy** by using only the most reliable sensors at each moment.

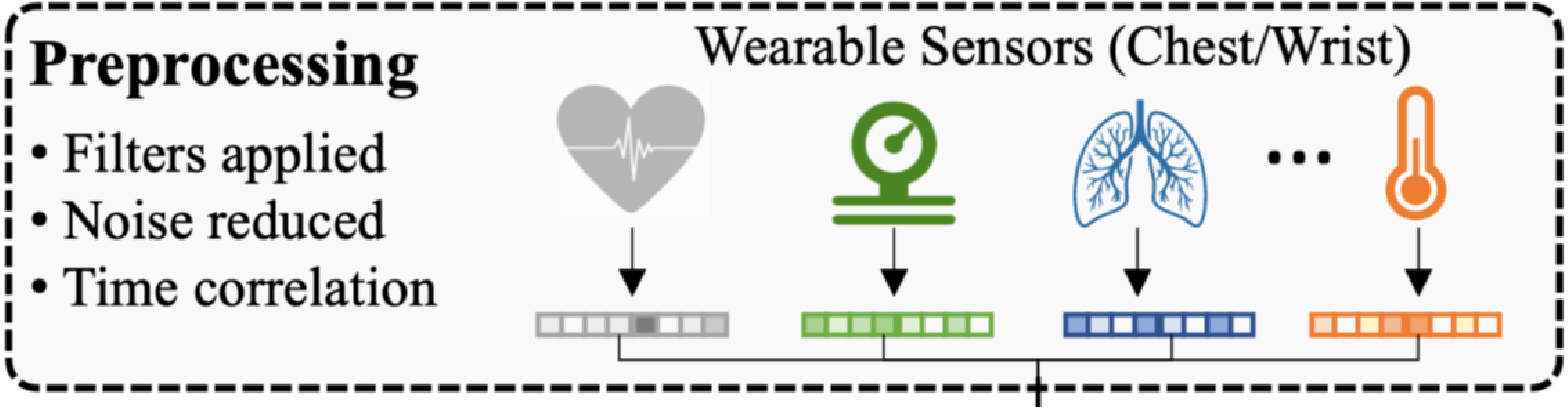
Overview of the SELF-CARE Framework

- **Goal:** Detect stress adaptively by selecting the most reliable sensors based on noise context.
- **Step 1 – Preprocessing:**
  - Filter raw signals to reduce noise and align data across sensors.
- **Step 2 – Context Identification:**
  - Extract features from motion (ACC) or muscle activity (EMG).
  - Gating model decides which sensor combinations (branches) to use.
- **Step 3 – Classification:**
  - Each branch (set of sensors) has its own ML classifier (Random Forest / AdaBoost).
- **Step 4 – Fusion via Kalman Filter:**
  - Combines outputs from selected branches.
  - Incorporates temporal dynamics for smoother, more accurate stress prediction.



Preprocessing – Signal Preparation

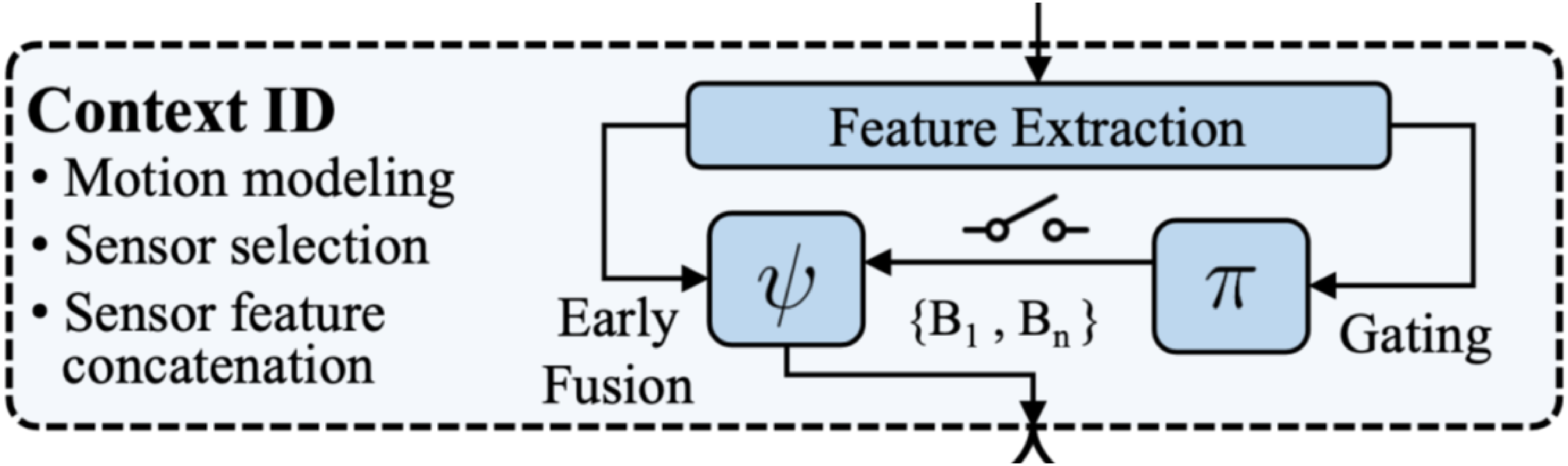
- **Removes noise** from raw physiological signals using filters (e.g., Butterworth, FIR, Savitzky-Golay).
- **Standardizes data** collected from multiple sensors (chest and wrist).
- **Segments signals** into fixed time windows (e.g., 60 s with 5 s overlap) for model input.
- **Enhances signal quality** so later modules can extract meaningful physiological features.
- Ensures all sensors are **time-aligned** for accurate feature correlation and fusion.



Context Identification

- **Goal:** Detect the current *noise context* (motion or muscle activity) to decide which sensors are reliable.

- **Input:** Raw sensor signals (ACC for wrist, EMG for chest).
- **Output:** The best sensor branch(es) to use for stress detection.
- **Main Components:**
  1. Feature Extraction
  2. Gating Model (π)
  3. Performance–Computation Trade-off (δ)
  4. Early Fusion (ψ)



Context Identification

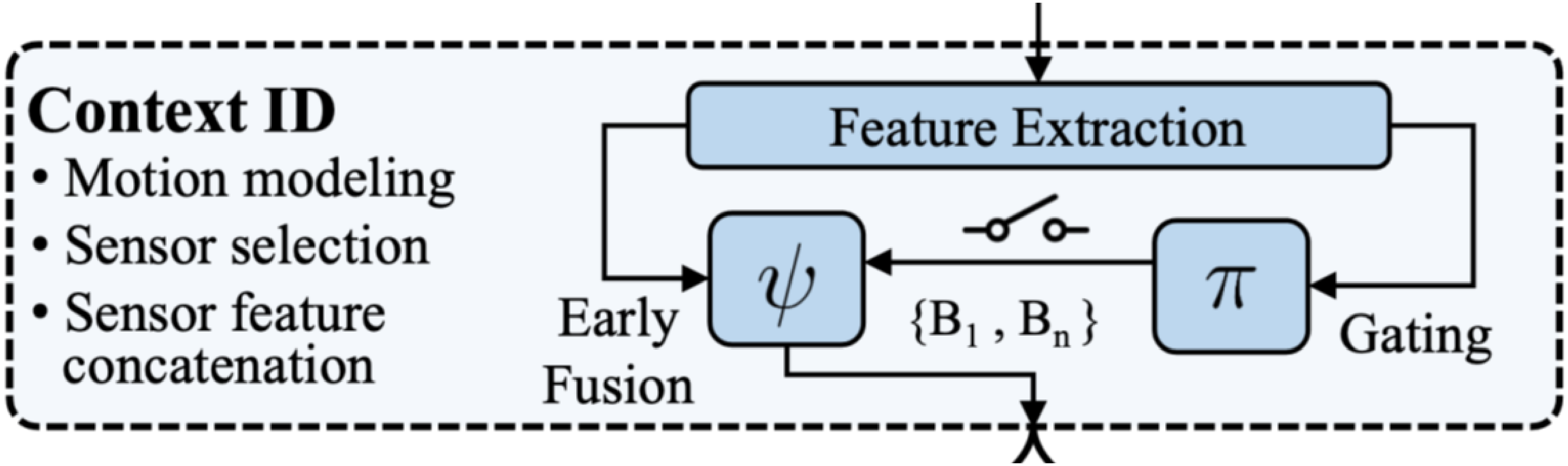
1. Feature Extraction

- Extracts **ACC (wrist)** or **EMG (chest)** features to capture motion or muscle activity.
- Models the **noise context**, not stress itself.
- Provides input to the **gating model** for sensor selection.
- Other sensor features are extracted **after** the gating decision.

2. Gating Model (π)

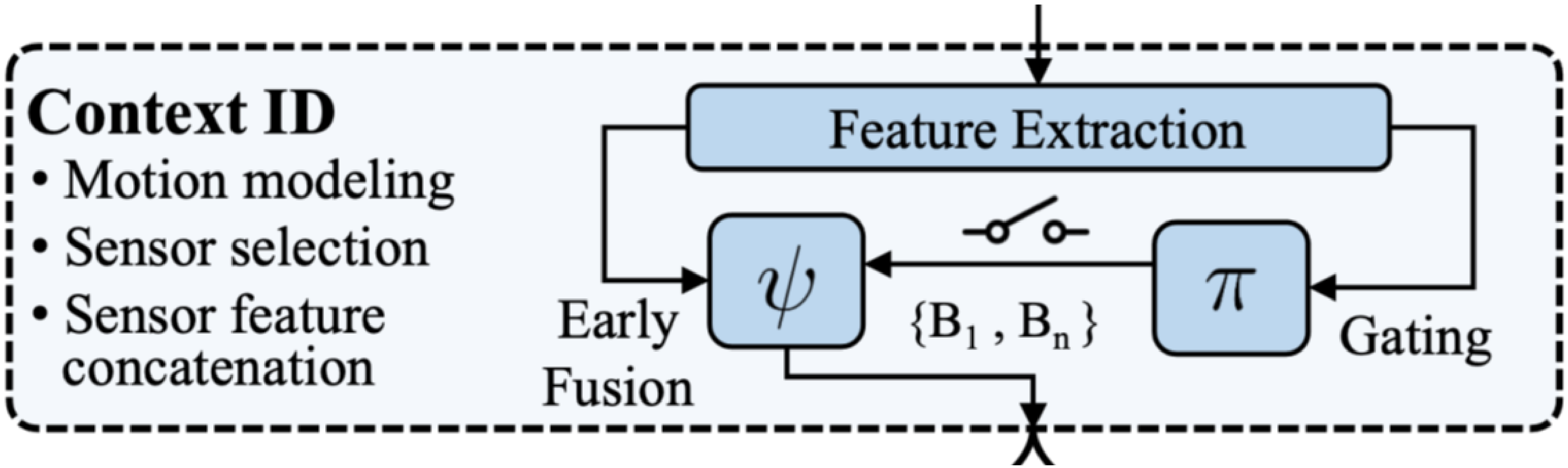
- **Decision Tree** predicts which **sensor branch** is most reliable.
- Uses **ACC/EMG features** as input.
- **Wrist**: chooses among 3 Random Forest branches (WB1, WB2, WB3).
- **Chest:** chooses among 5 AdaBoost branches (CB1, CB12, CB14, CB24, and CB27 for 3-class; CB5, CB7, CB9, CB13, and CB20for 2-class).

- **Lightweight & adaptive** — enables real-time context-aware selection.



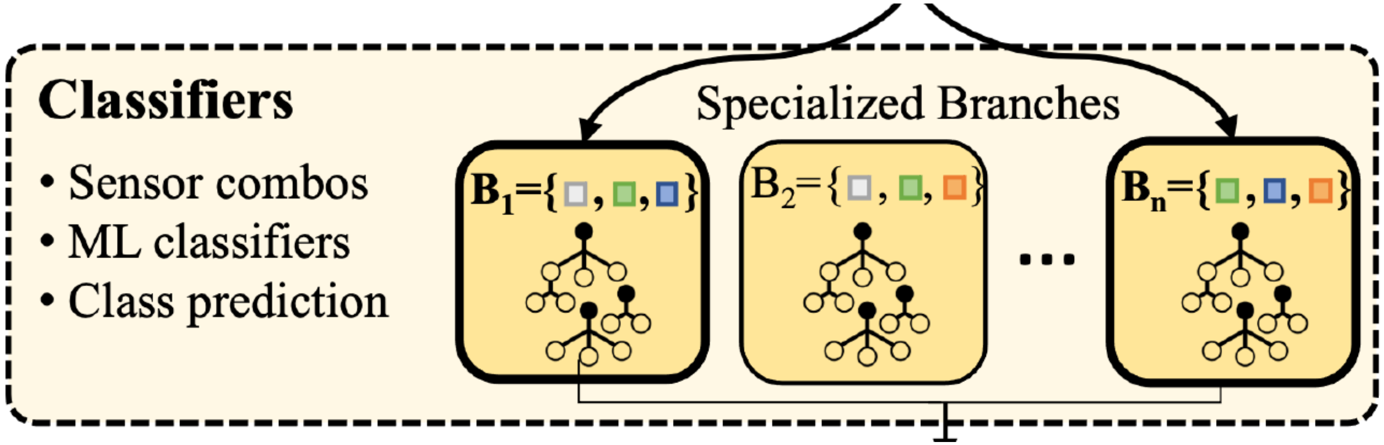
Context Identification

3. **Performance–Computation Trade-off (δ)**
   - Balances accuracy vs. device efficiency.
   - $\delta \in [0, 1]$ determines how many branches are selected:
     - **δ = 0:** only the top-probability branch (fast, low power).
     - **Higher δ:** more branches included (higher accuracy, more computation).
4. **Early Fusion (ψ)**
   - For each selected branch, features from its sensors are **concatenated** into a single vector.
   - These fused features are then passed to their **branch classifiers** (e.g., Random Forest, AdaBoost).



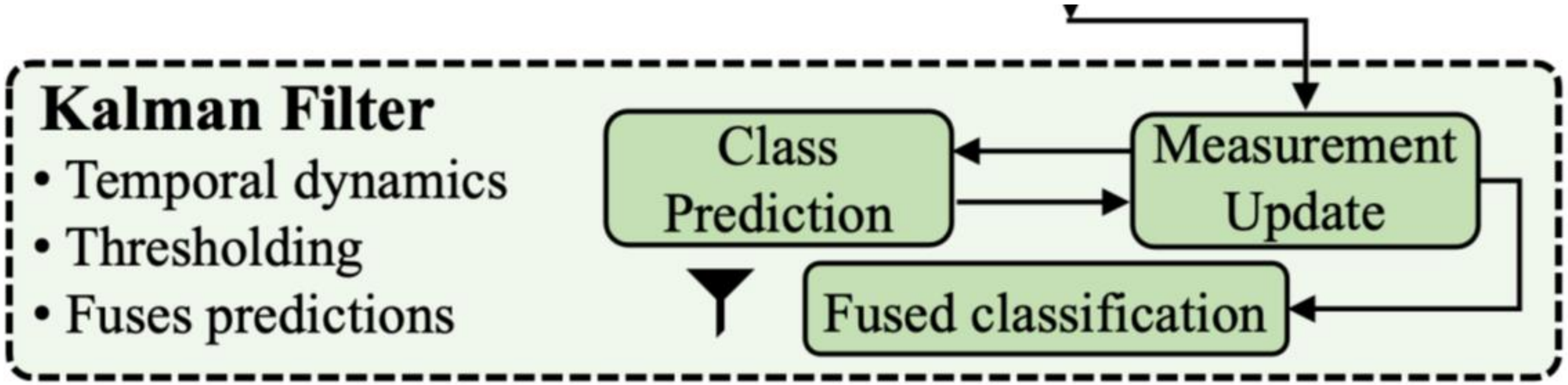
Branch Classifiers – Specialized Sensor Models

- Each **branch (B_1, B_2, …, B_n)** is a **separate classifier** trained on a specific **sensor combination**.
    - **Wrist devices:** use **Random Forest** classifiers.
    - **Chest devices:** use **AdaBoost** classifiers.
- Each branch predicts the **stress class** (baseline/ stress/ amusement).
- The **gating model** activates one or more branches based on the detected context.
- These outputs are later **fused** using the Kalman filter for the final stress prediction.



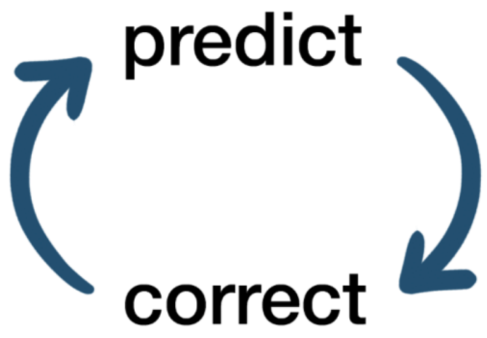
Late Fusion – Kalman Filter

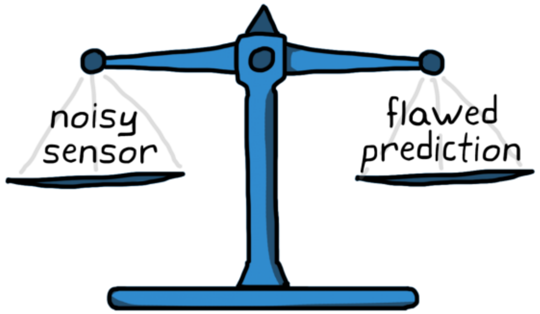
- Combines outputs from all **selected branch classifiers**.
- Uses **Kalman filtering** to model **temporal dynamics** — considers how stress levels evolve over time.
- Performs **prediction and measurement update** steps to refine class probabilities.
- Applies **thresholding** to handle noisy or uncertain predictions.
- Produces a **final fused classification** that is smoother and more accurate than simple voting.



Kalman Filter

- Used when we have **noisy or uncertain measurements**.
- It **predicts** what the next value should be (based on the past), then **updates** that prediction using the new data.
- Gives **more weight** to reliable readings and **less** to noisy ones.
- Produces a **smooth, realistic trend** instead of sudden jumps.





Result

- Traditional models fuse all sensors blindly, but SELF-CARE **selectively fuses** them based on **context** and uses **Kalman filtering** to smooth predictions, which leads to more stable and accurate stress detection.

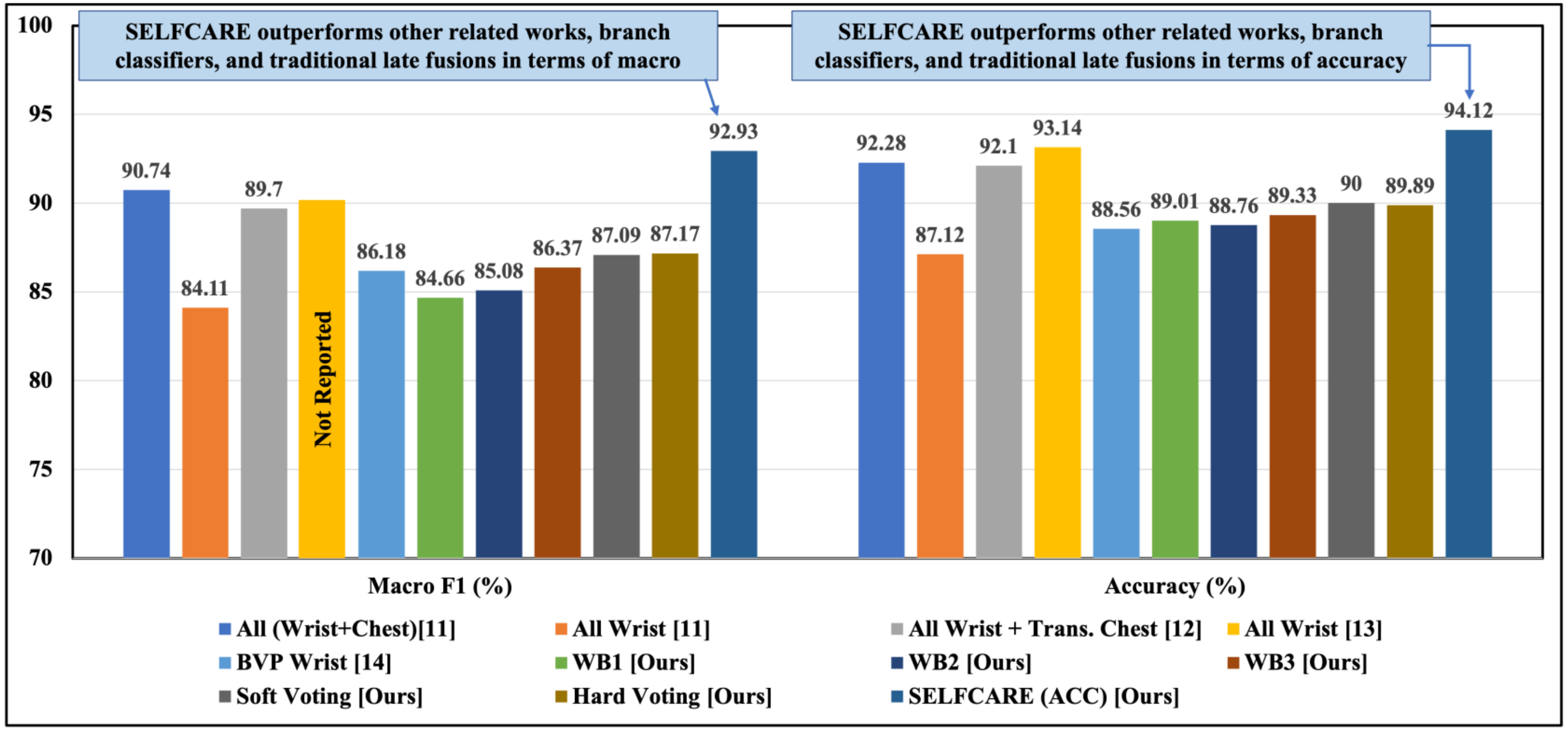


Fig. 5: Overall performance comparison of related works using LOSO validation on wrist data 2-Class. Results show that SELFCARE outperforms the related works, branch classifiers, and other traditional late fusion methods in terms of both macro F1 and accuracy.

- Thanks

Back –up slides

What happens before the Kalman filter

- After **context identification**, the **gating model** chooses which branches to activate — for example, **WB1 and WB2** for wrist sensors.
- Each **branch** is a **classifier** (like a Random Forest) that gives **probabilities** for each stress class.
  Let's assume it's a **3-class problem** → *Baseline, Stress, Amusement*.
- 🧠 **Example**
- If the gating model selects **WB1** and **WB2**, you get these predictions:

Branch	Baseline	Stress	Amusement
WB1	0.6	0.3	0.1
WB2	0.5	0.4	0.1

Each branch gives a **vector of probabilities**, like

$$Y_1 = [0.6, 0.3, 0.1], Y_2 = [0.5, 0.4, 0.1]$$

What the Kalman filter receives

- The **inputs (measurements)** to the Kalman filter are **these probability vectors** from all selected branches.
  So for each time segment (e.g., every 60 seconds of sensor data), the Kalman filter gets something like:

  - $z(k) = \{Y1, Y2, \ldots, Yn\}$

- where each Y_i is a probability vector from one branch.

What the Kalman filter does

- **Predict step:**
  - It predicts what the class probabilities *should be now* based on the previous time step.
  - Example: if at the last moment the final stress probabilities were [0.5, 0.4, 0.1], it expects something similar this time (stress doesn't change instantly).
- **Update step:**
  - It takes the new **branch outputs** (z(k)) and **updates** the prediction.
  - It gives **more weight** to branches that are more consistent with the previous state (less noise).
  - It gives **less weight** to sudden outliers or contradictory predictions.
- **Output (state):**
  - The Kalman filter produces the **fused, smoothed probability vector**:
- $x(k \mid k) = [P_{baseline}, P_{stress}, P_{amusement}]$

  - That becomes the **final stress prediction** for that time segment.

🔍 example (numerical)

- At time t_1
  - WB1 → [0.6, 0.3, 0.1]
  - WB2 → [0.5, 0.4, 0.1]
  - Kalman output → **[0.55, 0.35, 0.1]**

- At time t_2
  - WB1 → [0.1, 0.8, 0.1] (maybe noise spike)
  - WB2 → [0.4, 0.5, 0.1]
  - (it doesn't jump to 0.8 stress immediately)

  Kalman output → **[0.45, 0.45, 0.1]**
- At time t_3
  - WB1 → [0.2, 0.7, 0.1]
  - WB2 → [0.3, 0.6, 0.1]
  - Kalman output → **[0.35, 0.55, 0.1]** (gradually increasing — smooth transition)