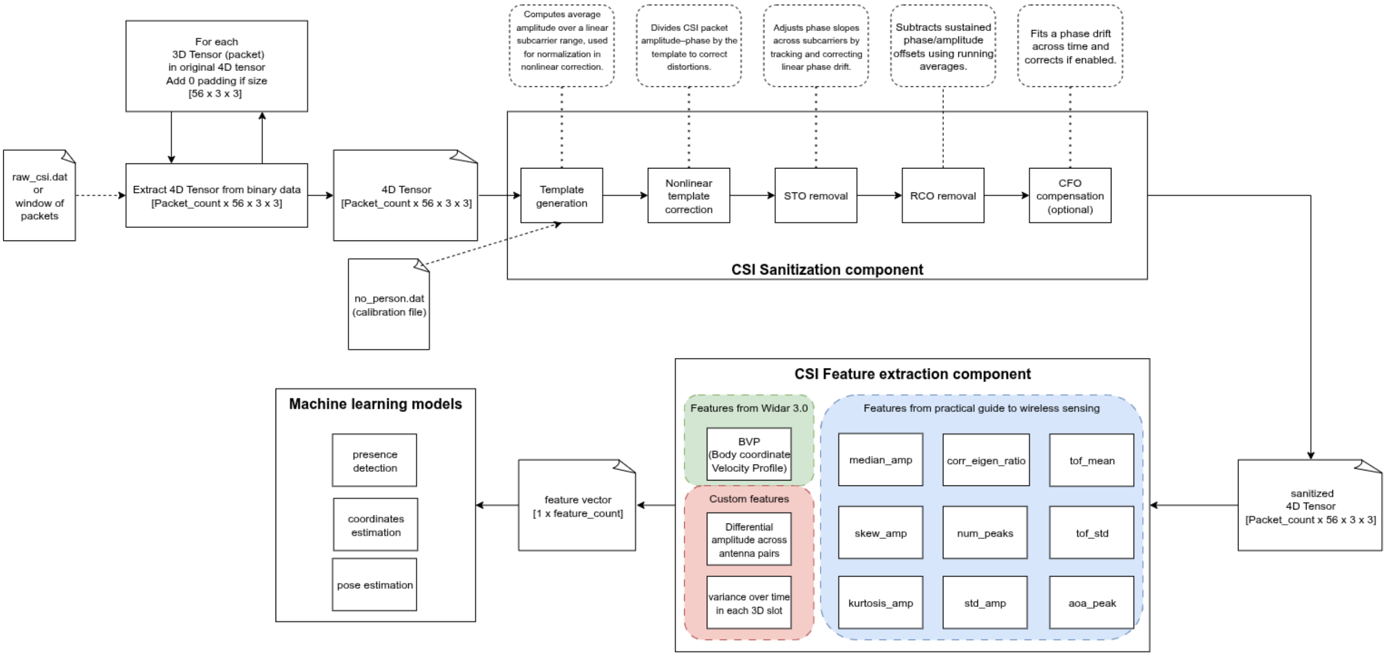
New Model Training Strategy

We need to train 3 models

1) Human presence
2) Coordinate pair (x, y)
3) Pose estimation



DATA format

CSI Data format
4D tensor = packet_count x 3 receiving antennas x 3 transmitting antennas x 56 subcarriers
Each value in the tensor is a complex number where the real part is amplitude and the imaginary part is phase.

In [the *Sanitization* section of the Practical Guide to Wireless Sensing](), the authors emphasize that: “Raw phase values in commodity Wi-Fi devices suffer from random offsets due to hardware impairments like STO, CFO, and RCO. Without calibration, using phase directly introduces **a lot of noise**."

They recommend **discarding unprocessed phase values** altogether unless:

- Proper sanitization is applied
- Or specific, robust transformations are used (e.g., **phase difference across subcarriers** or **temporal differencing**, not raw phase)

File name format

ilive-<pose>-<position number>-<orientation to routers>-<duration>.dat
ilive-standing-pos1-los2-1min.dat

CSI Data sanitization

Note: any packet with NaN/infinite values in the data files will be skipped during sanitization.

1. Template generation
   Purpose: To create a reference amplitude (baseline) for each Rx–Tx antenna pair by averaging the signal from a clean, empty-room recording.
   Input:
   - A 4D CSI tensor from **no_person.dat**: shape (packets, 56, 3, 3)
   - A linear_interval, typically subcarriers 20–39 (center of the spectrum)
   What happens:
   - Extract amplitude from each CSI packet: **np.abs(calib_csi[:, :, :, linear_interval])**
   - Take the **mean amplitude** over packets and selected subcarriers
   - Resulting shape: **(3, 3)** → one value per Tx–Rx antenna pair
   Output:
   - A **template** matrix (3×3) used to normalize future CSI packets
   Why?:
   This removes hardware and environment bias (e.g., static reflections, gain imbalance) before we compare new CSI data.

2. Nonlinear template correction
   Purpose:
   To remove hardware-induced **nonlinear distortions** in CSI amplitude and phase using the clean template.
   - CSI packet csi (shape: [packets, 56, 3, 3])
   - Template [3 x 3] matrix from Step 1
   What happens:
   - Compute **magnitude** and **unwrapped phase** of each CSI entry
   - Normalize amplitude: **amp / template**
   - Combine with phase: **amp_normalized × exp(1*j × phase)**
   Output:
   - A new **CSI tensor [packets, 56, 3, 3]** where distortions across frequency have been corrected.
   Why?:
   - Wireless hardware causes per-subcarrier inconsistencies that must be corrected before extracting features.

3. STO removal (Sampling Time Offset)
   Purpose:
   To remove linear phase slopes across subcarriers caused by small time misalignments between transmitter and receiver.
   Input:
   - CSI with corrected amplitude (shape: **[packets, 56, 3, 3]**)
   What happens:

- Compute the unwrapped phase of each subcarrier across 56 subcarriers
- Fit a line to phase (first to last subcarrier): slope = (φ_last - φ_first) / 55
- Subtract this linear trend to "flatten" the phase

Output:
- CSI with corrected phase slope for each Rx–Tx pair.

Why?
- Without STO correction, the CSI phase contains a strong linear trend that can mislead downstream models.

4. RCO removal
   Purpose:
   To remove residual offsets in amplitude and phase that remain even after STO correction.
   Input:
   - Flattened CSI signal from Step 3
   - Running average of **phase** and **amplitude** over previous packets (stateful)
   What happens:
   - (keeps track of the original offsets for use in calculations) For every packet in the file, compute:
     + the phase difference between the calibration template value and the packet.
     + the ratio between the amplitude of that packet and the amplitude of the calibration template at the 0th long training field (L = 1).
   - Rebuild the packet at that point with the amplitude ratio and the phase difference above in mind.

   Output:
   - Signal with stable amplitude and phase, cleaned from long-term drift
   Why?:
   - These residual offsets distort signal statistics and must be removed for consistency across time.

5. CFO compensation (optional)
   Purpose:
   To remove phase drift across time, i.e., over multiple packets. Usually relevant in long recordings.
   Input:
   - Phase of each packet's center subcarrier (e.g., subcarrier index 28)
   What happens:
   - Calculating the phase components of the first and second (and potentially more) LTF fields in the packet, and taking the phase difference of these two values.
   - Calculating the mean phase difference over all antennas that transmit that packet.
   - CFO = mean phase difference / time difference between 2 consecutive LTF fields. (since in this case, we are only considering the first 2 fields)

   Output:
   - CSI that remains **phase-aligned over time**
   Why?:

- In real-time streaming or long sessions, frequency differences between Tx and Rx cause accumulating phase errors.

Feature extraction in CSI-based sensing

What is a feature

A feature is an individual measurable property, variable, or attribute of the phenomenon being observed.

- In traditional ML, features are columns in a table (like age, height, income).
- In image ML, a feature might be pixel intensity.

Think of features as the *inputs* to your model that help it make decisions.

In the context of Wifi sensing, a feature is a derived statistic from a CSI window that helps describe a human's

1) presence
2) location
3) pose

Why extract features for wifi sensing?

Raw CSI data is high-dimensional, noisy, and variable — especially across time and environments. Machine learning models do not generalize well if they are fed raw complex CSI tensors directly.

Feature extraction addresses this by:
1) Reducing dimensionality and variability.
2) Emphasizing signal properties that correlate with physical events (e.g., human presence, pose).
3) Making learning more robust, interpretable, and transferable.

Feature properties in ML

Feature count

- Too few features: model may underfit (can't capture complexity).
- Too many features: model may overfit or learn spurious patterns.

Feature Quality

- Relevant features = high signal-to-noise ratio.
- Redundant or irrelevant features = noise that hurts performance.
- It's better to have fewer high-quality features than many low-quality ones.

Best Practices for Choosing Features

1. Domain knowledge first
    a. Use understanding of wireless signals (ToF, AoA, energy patterns).
    b. Don't blindly include all values from the CSI tensor.
2. Sanitize before extract
    a. Remove offsets and noise (STO, RCO, etc.) before feature calculation.
3. Aggregate over windows
    a. Use temporal windows (e.g., 60 packets) to smooth noise and variability.
4. Validate feature usefulness
    a. Use statistical analysis (e.g., correlation with labels).
    b. Try feature importance methods (`.feature_importances_`, SHAP, etc.)
5. Avoid data leakage
    a. Features must only use information *available at inference* time.
    b. Don't include future data in the training feature window.
6. Avoid high correlation between features
    a. Remove or combine redundant features (e.g., PCA, mutual information).

Feature list for Wifi sensing models

from paper

Feature name	Type	Meaning / Purpose	Calculation	Usage in model training
tof_mean	Physics-based	Mean delay from the peak of Channel Impulse Response (CIR) — approximates distance	FFT of CSI → magnitude of CIR → argmax index → mean of indices	ToF (Time of Flight) reflects distance; might be useful for position estimation, good for detecting large movements (presence), critical for coordinates.
tof_std	Physics-based	Variation of delays across antennas — hints at multipath spread or object extent	Same as above → std deviation of peak indices	ToF spread captures multipath richness — larger spreads may indicate complex movements or poses.
aoa_peak	Spatial / AoA	Estimated angle of arrival where signal power is strongest — approximates direction	MUSIC algorithm on antenna covariance matrix → angle with max power	AoA changes with location but can be noisy with pose; useful for precise spatial info. Not very useful for human detection.
corr_eigen_ratio	Spatial correlation	Eigenvalue ratio of CSI correlation matrix — indicates signal richness or DoF	Largest eigenvalue / sum of eigenvalues of correlation matrix	Eigen-ratio indicates signal richness — rich multipath = person present or moving in complex ways. (not really useful for coordinates)
num_peaks	Temporal frequency	Number of signal energy peaks — high motion or multipath	Smooth magnitude of CSI (1D) → scipy.find_peaks	More peaks indicate movement or shape complexity; too unstable for exact coordinates.

		yields more peaks		
std_amp	Statistical amplitude	Spread of signal energy — useful for presence detection	Std deviation of smoothed magnitude	Amplitude variance changes with movement intensity or body shape. Not useful for coordinates
median_amp	Statistical amplitude	Median energy — helps normalize data or track changes in baseline	Median of smoothed magnitude	Baseline signal energy; good for detecting presence but not informative alone. Doesn't give any special info.
skew_amp	Shape descriptor	Asymmetry of energy distribution — helps distinguish poses or irregular movement	Skewness of smoothed amplitude distribution (scipy.stats.skew)	Skewed distributions reveal leaning, crouching or asymmetric motion (pose estimation) but useless for coordinates.
kurtosis_amp	Shape descriptor	Peakedness of energy distribution — detects sharp changes or sudden movement	Kurtosis of smoothed amplitude (scipy.stats.kurtosis)	High kurtosis = sharp/sudden pose → crouch vs. stand → highly relevant. But useless for coordinates

ToF-based features (`tof_mean`, `tof_std`)

1) Compute CIR (Channel Impulse Response):
   `CIR = np.fft.ifft(csi_packet, axis=0)`
2) Then take the magnitude:
   `cir_abs = np.abs(CIR)`
3) For each antenna pair, find the index of max energy (delay peak).
4) Use `np.mean()` and `np.std()` over all antenna pairs.

AoA / Direction-of-Arrival (aoa_peak)

Use MUSIC algorithm approximation:

1) Compute antenna correlation matrix R
2) Eigen-decompose to get signal + noise subspace
3) Use steering vector scan over angles `-90° to +90°`
4) Find peak in spatial spectrum = estimated AoA

Correlation Eigenvalue Ratio (corr_eigen_ratio)

1) Extract average CSI across subcarriers (i.e., per antenna)
2) Build correlation matrix of this data
3) Compute eigenvalues
4) Compute:
   `eig_vals[0] / np.sum(eig_vals)`
   where `eig_vals[0]` is the largest eigenvalue

Amplitude Features (`std_amp`, `median_amp`, `skew_amp`, `kurtosis_amp`)

1) Extract 1D signal:
`signal_1d_amp = np.abs(csi_packet[:, 0, 0])`

2) Apply smoothing:
`gaussian_filter1d(signal_1d_amp, sigma=1)`

3) Then compute:

- `std`: measures spread
- `median`: central tendency
- `skew`: left/right asymmetry
- `kurtosis`: how sharp/flat the peak is

Number of peaks num_peaks

- Apply peak detection on smoothed 1D CSI magnitude
- Use scipy.signal.find_peaks
- Count number of peaks above mean level

Why these features?

These features are:

- Lightweight to compute
- Robust to small noise
- Meaningful in physical sense (e.g., delay, AoA)
- Reusable across presence, pose, localization tasks (but it's important to choose right features for each model)

They're used as inputs to:

- Traditional ML models (e.g., XGBoost, LightGBM)
- Or fed into deep learning pipelines as compact representations

Extra features (not from practical guide paper)

Consider **hand-engineering additional features** like:

1. **Variance over time in each 3D slot**
   Usage: Human presence, Pose estimation
   We compute **CSI magnitude variance** across a **window of CSI packets (60)** at each (rx, tx) pair → it reflects **temporal signal instability** at each spatial link.
   Why it's useful:
   - High variance: likely motion (e.g. person present or moving)
   - Low variance: stillness or no person

2. **Differential amplitude across antenna pairs**
   Usage: Pose estimation, (x, y) coordinates
   Captures the **imbalance** in signal strength across antenna pairs.

Movement or occlusion changes this imbalance.
Why It's Useful:
- Used for pose recognition and localization.
- Some poses reflect/absorb signals unequally → spatial asymmetry.

Extra features from widar 3.0 paper

BVP (Body-coordinate Velocity Profile) is a robust feature derived from CSI data that captures **how human body parts are moving**—in terms of their speed—**regardless of their location or orientation**.

It works by:
1) converting time-domain CSI amplitude/phase changes into **velocity information** using Doppler shift (via FFT)
2) then computing a velocity profile (power spectrum over Doppler bins) that represents **movement intensity at different velocities.**

Why it's useful
1) Human Presence - Moving bodies cause clear Doppler shifts; no movement = flat BVP spectrum
2) Coordinate Estimation - Localized motion energy may correlate with **position-dependent reflections. (Probably wouldn't work at all, so ignore it)**
3) Pose Estimation - Each pose has unique velocity distribution over time (e.g., crouching vs walking)

Feature selection based on a model

One of the assumptions is that not all features are equally informative for every task. It's not a good idea to feed 9 features to each of the models, especially because some of the features just from a physical standpoint cannot be used for certain calculations.

Below is a task-specific breakdown of the 9 features we extract, explaining why each feature is useful for 3 models.

Human presence detection model

Goal: binary classification (is there a human?)
Use features that capture **signal strength, variation, and distortion**:
['tof_mean', 'tof_std', 'corr_eigen_ratio', 'num_peaks',
 'std_amp', 'median_amp', 'skew_amp', 'kurtosis_amp']
Drop: aoa_peak → not stable when a person is stationary but present.

Coordinates (x, y) estimation

Goal: regression to (x, y)
Use features that are **directly linked to spatial location**:
['tof_mean', 'tof_std', 'aoa_peak', 'corr_eigen_ratio']

Drop: num_peaks, std_amp, skew_amp, etc. — these are pose/motion-dependent and can mislead position models.

Pose estimation

Goal: classify poses (e.g., standing, crouching)
Use features describing **signal distortion and shape**: ['tof_std', 'corr_eigen_ratio', 'num_peaks', 'std_amp', 'skew_amp', 'kurtosis_amp']
Drop: ‘tof_mean’ and ‘aoa_peak’: these vary with position but don’t correlate directly with pose.

Best practices

1. Maintain a separate feature selection pipeline for each task.
2. Use tools like:
    a. sklearn.feature_selection.SelectKBest
    b. SHAP for feature importance
    c. pycaret built-in feature ranking
3. Optionally use **PCA** to reduce the feature set if too many are correlated.