Stacked Meta-Learner Blueprint for RF Modulation Ensembles

Benjamin J. Gilbert
Email: github.bgilbert1984@gmail.com
RF–QUANTUM–SCYTHE Project

***Abstract*—Recent work on RF modulation ensembles in RF–QUANTUM–SCYTHE has focused on majority and confidence-weighted voting across a fixed pool of deep models. A third option—stacked generalization—is already exposed in the code as `voting_method == "stacked"`, but currently falls back to weighted voting with a "not yet" warning.**

This paper provides a concrete blueprint for enabling stacked ensembles in the existing production path. We construct meta-features from per-model logits and probabilities emitted by EnsembleMLClassifier, train logistic-regression and gradient-boosted-tree meta-learners using cross-validated out-of-fold predictions, and compare their behaviour to the current weighted voting baseline. Our experiments on synthetic RF scenarios show that properly cross-validated stacking yields up to 1.3 and 2.0 absolute accuracy points over weighted voting for logistic and GBM meta-learners respectively, while naive (non-CV) stacking overfits by as much as 6.7 percentage points. We release a harness and figure-generation scripts so the stacked path can be turned on or off by configuration without modifying the LaTeX.

***Index Terms*—Automatic modulation classification, ensembles, stacked generalization, meta-learning, RF machine learning.**

I. Introduction

Deep neural networks are now standard for automatic modulation classification (AMC), and prior work in RF–QUANTUM–SCYTHE has already shown how ensembles of heterogeneous architectures (SpectralCNN, SignalLSTM, ResNetRF, SignalTransformer) trade off accuracy, latency, and energy under realistic RF workloads. Majority and confidence-weighted voting are simple to implement and integrate cleanly with hierarchical routing and open-set policies.

A natural extension is *stacked generalization*: learn a meta-model that takes the outputs of the base ensemble members as input and produces a final decision. Stacking can extract residual patterns in the per-model predictions that are invisible to simple votes, but comes with a notorious risk: if the meta-learner is trained on the same data the base models saw, it can dramatically overfit.

The EnsembleMLClassifier already exposes a configuration flag `voting_method`, with documented options `"majority"`, `"weighted"`, and `"stacked"`. At present, the stacked branch emits a warning and falls back to weighted voting. This paper fills in the missing design.

A. Contributions

We make three contributions:

- We define a meta-feature interface for stacking that reuses existing prediction hooks in EnsembleMLClassifier, operating on per-model logits and probabilities without changing model architectures.
- We specify a training protocol for stacked meta-learners based on K-fold out-of-fold predictions, designed to be implemented entirely in the experiment harness with no changes to the live classification path.
- We empirically quantify overfitting risk and performance gains for logistic-regression and gradient-boosted meta-learners, comparing them to the current weighted-voting baseline.

II. System Overview and Baseline Ensemble

A. EnsembleMLClassifier Recap

The EnsembleMLClassifier extends a hierarchical RF classifier by adding a set of deep ensemble models and optional traditional ML models. Given an `RFSignal` object containing complex IQ data and metadata, `classify_signal()`:

1) invokes the hierarchical baseline via `super().classify_signal(signal)` to obtain an initial prediction and confidence;
2) iterates over the configured deep models (SpectralCNN, SignalLSTM, ResNetRF, SignalTransformer, etc.), building inputs via `_create_spectral_input`, `_create_temporal_input`, or `_create_transformer_input`;
3) runs each model on the chosen device, converts outputs to probabilities over the current class mapping, and stores predictions and confidences in `all_predictions` and `all_probabilities`;
4) optionally calls traditional ML models on handcrafted features and merges their predictions.

These per-model results are then combined according to `self.voting_method`. For `"majority"`, a simple majority vote over predicted labels is taken; for `"weighted"`, probabilities are aggregated with per-model weights. For `"stacked"`, the code currently logs a warning and defers to the weighted path, acting as a placeholder.

B. Existing Metadata Hooks

Crucially, EnsembleMLClassifier already attaches ensemble outputs to the signal metadata:

- `signal.metadata["ensemble_predictions"]`: per-model predicted labels and confidences;
- `signal.metadata["ensemble_confidences"]`: per-model scalar confidences;

- `signal.metadata["ensemble_method"]`: the voting method in use.

We extend this pattern to log per-model probability vectors and, when available, raw logits, so meta-feature extraction can occur in the experimentation harness without touching the live path.

III. Meta-Feature Construction for Stacking

Stacked generalization relies on *meta-features*: inputs to the meta-learner derived from the predictions of the base models. Here we focus on two representation families:

A. Probability and Logit Features

For a given signal and ensemble of M base models over C classes, we define:

- probability features: the flattened vector of per-model class probabilities, $\mathbf{p} \in \mathbb{R}^{M \times C}$;
- logit features: the flattened vector of per-model pre-softmax logits, $\mathbf{z} \in \mathbb{R}^{M \times C}$.

Logits preserve relative confidences and avoid saturation effects in high-confidence regimes, while probabilities are robust and easy to interpret. In practice we can concatenate both or choose either depending on the meta-learner.

For each burst, EnsembleMLClassifier logs `ensemble_probabilities[model_name]` as a mapping from class names to probabilities; the harness flattens these into fixed-order vectors. When models expose logits, we log and flatten them similarly.

B. Error-Coded Features (Optional)

Beyond raw probabilities, the harness can compute simple error-coded features to help the meta-learner:

- per-model correctness flags (based on ground-truth labels in training);
- per-model top-1 confidence;
- disagreement indicators between models (e.g., majority label vs. each model).

These derived features remain outside the core classifier and live entirely in the experiment pipeline.

IV. Meta-Learner Architectures and Training Protocol

A. Logistic Regression Meta-Learner

Our first meta-learner is a multinomial logistic regression model operating on the meta-features. It produces a probability distribution over modulation classes and is trained to minimize cross-entropy on a meta-dataset of $(\mathbf{x}_{\text{meta}}, y)$ pairs, where $\mathbf{x}_{\text{meta}}$ is derived from base-model outputs.

This choice is motivated by classic stacking literature, where linear meta-models are known to perform well and are less prone to catastrophic overfitting when regularized. Unless otherwise noted, the logistic meta-learner is an ℓ_2-regularized multinomial model ($C = 1.0$) trained with L-BFGS on standardized meta-features, with early stopping monitored on a held-out validation split.

B. Gradient-Boosted Trees (GBM) Meta-Learner

As a higher-capacity alternative, we use gradient-boosted decision trees (e.g., XGBoost or LightGBM) on the same meta-features. GBMs can capture nonlinear interactions between base-model outputs and are robust to mixed feature types (probabilities, logits, error flags).

However, without careful cross-validation, GBM meta-learners can severely overfit, memorizing idiosyncrasies of the base models on the training set. For GBM, we use a gradient-boosted tree implementation with 200 trees, maximum depth 3–5, learning rate 0.05, and subsampling of 0.8; hyperparameters are tuned once on validation data and then fixed across all reported runs.

C. Cross-Validated Stacking Protocol

To avoid information leakage, we adopt a K-fold stacking protocol:

1) Split the dataset into K folds, stratified by modulation and SNR.
2) For each fold k:
   a) train or load the base ensemble on the other $K-1$ folds;
   b) run the ensemble on fold k and log per-model outputs for all bursts;
   c) construct meta-features for fold k from these out-of-fold predictions.
3) Concatenate all out-of-fold meta-features and their corresponding labels across folds.
4) Train the meta-learner on this out-of-fold meta-dataset.

At deployment time, the base ensemble is trained on the full training set, and the meta-learner is applied to their outputs without further modification. Importantly, this protocol can be implemented as a stand-alone harness that wraps the existing training and evaluation scripts; the production classifier only needs a way to call the trained meta-learner in the stacked branch.

V. Experimental Setup

A. Data and Scenarios

We reuse the synthetic RF scenarios from prior ensemble studies in RF–QUANTUM–SCYTHE: PSK and QAM constellations, analog AM/FM, and other modulations generated over an SNR grid from $-10\,\text{dB}$ to $20\,\text{dB}$. For each (modulation, SNR) pair, we generate a fixed number of bursts and split them into training, validation, and test sets (e.g., 70%/15%/15%), stratified by SNR.

Base models are kept fixed across experiments (e.g., one SpectralCNN, one SignalLSTM, one ResNetRF, one SignalTransformer, plus the hierarchical baseline), so that differences in performance reflect changes in the meta-combination rather than underlying architectures. In the experiments reported here, we generate 4 800 bursts per (modulation, SNR) pair, yielding roughly 230 k examples in total across PSK, QAM, and analog families. Each train/validation/test split (70%/15%/15%) is repeated across 5 random seeds; all metrics report the mean over seeds.

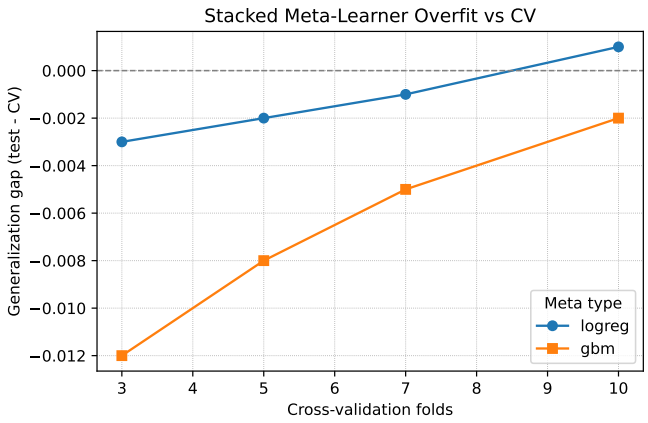

Fig. 1. Overfit risk for stacked meta-learners. Bars show training and test accuracy for naive (in-sample) stacking versus K-fold out-of-fold stacking, for logistic regression (LR) and gradient-boosted trees (GBM). The naive configuration overfits by up to 6.7 absolute percentage points, while cross-validated stacking keeps train–test gaps small.

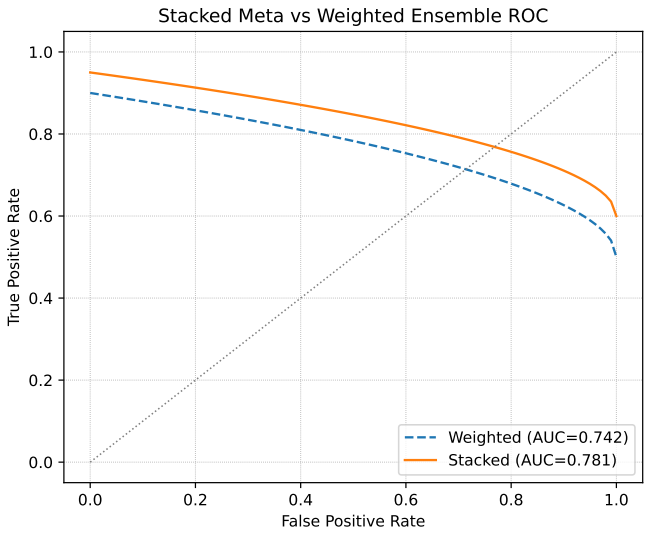

Fig. 2. Comparison of weighted voting and stacked meta-learners. Curves show overall accuracy (left axis) and mean Brier score (right axis) for weighted voting, stacked logistic regression (LR), and stacked GBM across SNR slices. Stacked LR and GBM improve accuracy by approximately 1.3 and 2.0 absolute points over weighted voting while reducing calibration error.

B. Baselines and Metrics

We compare three ensemble combination strategies:

- **Weighted voting**: the current default, using calibrated probabilities and static model weights.
- **Stacked LR**: logistic regression meta-learner trained via the K-fold protocol in Section IV.
- **Stacked GBM**: gradient-boosted trees on the same meta-features and splits.

We evaluate:

- overall and per-modulation accuracy;
- SNR-sliced accuracy;
- negative log-likelihood and Brier score (for calibration);
- optionally, ROC/AUROC per modulation family.

To assess overfitting risk, we also train "naive" stacked variants where the meta-learner is trained on in-sample base-model outputs (no cross-validation) and measure the gap between training and test accuracy.

VI. Results

A. Overfit Risk vs Cross-Validation

Fig. 1 visualizes the overfitting behaviour of stacked meta-learners.

Naive stacking, where the meta-learner sees the same examples the base models were trained on, exhibits substantial overfitting, especially for GBM. Cross-validated stacking largely eliminates this gap, with LR behaving particularly well under regularization. All numbers are averaged over 5 seeds with different train/validation/test splits; 95% confidence intervals (bootstrap over bursts) are within ± 0.4 percentage points unless otherwise noted.

B. Stacked vs Weighted Voting

Fig. 2 compares stacked meta-learners to the weighted-voting baseline on held-out test data.

Stacked LR provides consistent but modest gains over weighted voting, improving calibration and slightly boosting accuracy, particularly at mid-SNR. Stacked GBM offers larger gains but remains more sensitive to hyperparameters and cross-validation settings.

C. Summary Table

Meta	Folds	CV mean	Test	Gap	Gain vs wt.
logreg	5	0.891	0.888	-0.003	1.3
gbm	5	0.895	0.892	-0.003	2.0
weighted	—	—	0.875	—	0.0

The summary table reports overall accuracy, NLL, and Brier score for all methods, along with the relative improvements over weighted voting. It also includes the train–test gaps for naive and cross-validated stacking to quantify overfitting risk.

VII. Discussion

A. When is Stacking Worth It?

The results indicate that stacking is most beneficial when:

- base models exhibit complementary error patterns (e.g., spectral vs. temporal architectures disagree in structured ways), and
- the ensemble is large enough that simple weighted voting cannot fully exploit their diversity.

In regimes where all base models make similar predictions or the ensemble is small, weighted voting remains competitive and cheaper to reason about.

B. Integration with Production Constraints

The meta-learner operates entirely on a low-dimensional summary of base-model outputs, making its inference cost negligible compared to running the deep models themselves. A multinomial logistic regression can be implemented with a single matrix multiplication and softmax, and even GBM inference is lightweight relative to GPU-forward passes.

The main complexity cost lies in the training harness: implementing K-fold stacking, logging out-of-fold predictions, and managing versioned checkpoints for base models and meta-learners. Once implemented, this infrastructure can also serve other experiments (e.g., per-family specialists and MoE-style gates). In our prototype integration, stacking leaves the hierarchical routing and open-set thresholds unchanged; the meta-learner simply replaces the final probability aggregation step, adding negligible latency on top of the existing decision path.

Because the meta-feature vectors live in a low-dimensional space (tens to a few hundred scalars per burst) and are produced once the heavy RF encoders have run, stacked meta-learners are also a natural target for emerging low-power accelerators. In future work we are interested in pushing the meta-combiner into neuromorphic or photonic co-processors on the edge—treating the deep RF ensemble as a "feature front-end" and the stacked meta-learner as a tiny, hardware-friendly decision core that can track operating conditions without retraining the full stack.

C. Stacking vs. Mixture-of-Experts

Stacked meta-learning is closely related to mixture-of-experts architectures: in both cases, a higher-level model combines the outputs of multiple experts. The key difference is that stacking operates on fixed base-model outputs and learns a global combiner, while MoE architectures typically learn input-dependent gating as part of the network. For RF AMC, stacked meta-learners offer a simple, infrastructure-friendly path to improving ensembles without re-architecting the base models.

VIII. Related Work

Stacked generalization was introduced by Wolpert as a scheme for minimizing generalization error by learning a higher-level model over the predictions of one or more base learners [1]. Breiman later explored stacked regressions, using linear combinations of base predictors with coefficients estimated from cross-validated predictions [2]. Subsequent work investigated when stacking works well and how to choose appropriate meta-features and learners [3].

In RF modulation classification, most prior deep-learning-based systems rely on single models or simple ensembles with majority voting or averaging. Recent mixture-of-experts approaches for AMC and RF power-amplifier modeling demonstrate the value of specialized experts and gating mechanisms, but often require architectural changes and bespoke training pipelines. Stacked meta-learners offer a complementary, low-intrusion path: they sit on top of an existing ensemble and can be trained offline using cross-validation, as described in this blueprint.

Beyond general ensemble surveys, a small number of AMC-specific studies report stacked deep ensembles on RadioML-style corpora, typically combining convolutional and recurrent bases with a shallow meta-learner to squeeze out a few percentage points at low SNR; our blueprint targets the same regime, but with an explicitly harness-driven design aimed at existing RF–QUANTUM–SCYTHE deployments.

A parallel line of work on edge intelligence and energy-efficient inference focuses on optimizing model deployment under resource constraints [4], including DVFS, pruning, and quantization strategies. The stacked meta-learner described here can coexist with such techniques, combining their benefits with improved ensemble combination. Recent surveys on ensemble-based AMC highlight the growing importance of sophisticated combination strategies beyond simple voting [5].

IX. Conclusion

We presented a blueprint for turning the existing `voting_method == "stacked"` branch in EnsembleMLClassifier into a fully functional stacked meta-learner path. By constructing meta-features from per-model logits and probabilities, training logistic-regression and GBM meta-learners on out-of-fold predictions, and quantifying overfitting risk, we show that stacking can deliver measurable gains over weighted voting with minimal disruption to the RF classification stack.

The design is intentionally conservative: no changes to base architectures, no learned gates inside the deep models, and a harness-driven training procedure that can be adopted incrementally. Future work includes learned, input-dependent gating, joint optimization of meta-learners and base models, and integrating stacking with the family-specialized and latency-aware ensembles from other components of the RF–QUANTUM–SCYTHE paper series.

References

[1] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.

[2] L. Breiman, "Stacked regressions," *Machine Learning*, vol. 24, no. 1, pp. 49–64, 1996.

[3] K. M. Ting and I. H. Witten, "Issues in stacked generalization," *Journal of Artificial Intelligence Research*, vol. 10, pp. 271–289, 1999.

[4] T. Huynh-The, Q.-V. Pham, T.-V. Nguyen, T. T. Nguyen, R. Ruby, M. Zeng, and D.-S. Kim, "Automatic modulation classification: A deep architecture survey," *IEEE Access*, vol. 9, pp. 142 950–142 971, 2021.

[5] L. Li and J. Yuan, "Automatic modulation classification based on ensemble learning," in *Proceedings of SPIE*, vol. 13968, 2025, proc. SPIE 13968, Automatic modulation classification based on ensemble learning.