**1. Data‐splitting & risk of data leakage**

1. **Single split vs. nested CV**
   * Right now you fit on 100% encoding and evaluate once on delay. That gives you one point estimate but no confidence interval.
   * **Recommendation:** Use a nested cross‐validation on encoding to both pick hyperparameters *and* quantify variability (e.g. 5-fold CV over encoding trials). Then train on all encoding and test on delay.
2. **No independent “encoding hold-out”**
   * Your current “va\_enc” is identical to train, so it adds no information. If you need within-epoch generalization, hold out a random 20% of encoding trials *before* you ever touch Pipe.fit.
3. **Stratification**
   * Ensure your splits (both within encoding and when you do a train/test split) are stratified on the class label. Unbalanced class splits can inflate train/test discrepancies.

**2. Hyperparameter tuning**

1. **Ridge α only**
   * You rely solely on RidgeClassifierCV(alphas=…) for tuning α. That’s fine, but you never validate MiniRocket’s own settings (num\_kernels, dilation ranges, bias term, etc.).
   * **Recommendation:** If you really care about kernel count or random seed, wrap your whole pipeline in a GridSearchCV or Bayesian optimizer *inside* the encoding CV.
2. **Fixed random\_state**
   * A single seed can hide variance. Consider repeating the entire pipeline with 3–5 different random\_state values and averaging results (or reporting median + IQR).

**3. Feature extraction & windowing**

1. **Raw voltage vs. baseline-corrected power**
   * MiniRocket works on raw series, but in iEEG you often want baseline normalization (e.g. z-score each trial relative to pre-stimulus).
   * **Recommendation:** For each trial subtract the mean & divide by SD of the fixation window before feeding into extract\_window. This removes slow drifts and aligns power scales.
2. **Window alignment**
   * You’re taking 0–1500 ms relative to encoding start as your encoding window. But if encoding duration varies across trials (e.g. jitter), you may be mixing different cognitive stages.
   * **Recommendation:** Use fixed event markers (e.g. stimulus onset, not trial-info[0]) and consider shorter, overlapping sliding windows (e.g. 200 ms stepped by 50 ms) to capture temporal dynamics.

**4. Model evaluation & metrics**

1. **Accuracy only**
   * Overall accuracy conflates class imbalance. You do compute TPR/TNR per class for the delay window, but not for fixation or encoding.
   * **Recommendation:** Report per‐class precision, recall, F1 and macro-averaged metrics for *all* windows.
2. **Statistical significance & multiple comparisons**
   * You’ll test many channels × subjects. If you test significance per channel, you must correct for multiple comparisons (e.g. false-discovery rate across channels or cluster‐based permutation tests).
   * **Recommendation:** Use a non‐parametric permutation test: shuffle trial labels 1,000× to build a null distribution of delay-accuracy, then compute a p-value. Correct across channels with FDR or cluster correction.
3. **Effect size**
   * Accuracy differences (e.g. encoding = 80%, delay = 60%) are more informative when you compute Cohen’s *d* or AUROC difference rather than raw percentages.

**5. Per-subject vs. group inference**

1. **Within-subject statistics**
   * You currently treat each channel as an independent datapoint. But channels from the same subject are correlated.
   * **Recommendation:** After obtaining per‐channel delay-accuracy, aggregate at the ROI level, then run a mixed-effects model (channels nested in subjects) to assess whether delay-accuracy is significantly above chance across subjects.
2. **Leave-one-subject-out (LOSO)**
   * If you ever extend to cross-subject decoding, use LOSO CV to avoid subject-specific overfitting.

**6. iEEG-specific preprocessing & artifact control**

1. **Line noise & referencing**
   * You’ve notch-filtered and median‐referenced in preprocessing, but your classifier will still pick up residual line noise artifacts if they vary across conditions.
   * **Recommendation:** After notch, apply a broadband high-pass (e.g. 1 Hz) and low-pass (200 Hz) filter. Consider common average referencing (CAR) vs. bipolar referencing and compare decoding performance.
2. **Artifact rejection**
   * Trials with artifacts (muscle bursts, blinks) can produce spuriously high classification.
   * **Recommendation:** Quantify trial‐by‐trial variance or amplitude threshold to drop artifacts *before* extract\_window, or incorporate an additional artifact‐detection model.

**7. Alternative decoding strategies**

1. **Temporal generalization**
   * Rather than three fixed windows, train on every timepoint (or window) and test on every other timepoint. This yields a time×time generalization matrix, revealing when encoding information re-emerges during delay.
2. **Dimensionality reduction**
   * MiniRocket is powerful but blind to physiologically‐meaningful features. Try simpler baseline models (e.g. LDA on band-power features) to see whether the complexity is warranted.
3. **Comparisons to chance**
   * Always benchmark against a “shuffle” model: run the exact same pipeline but with random class labels to confirm you’re above permutation chance.

**8. Code organization & reproducibility**

1. **Pipeline encapsulation**
   * Wrap your extract\_window → pipe.fit → pipe.score logic in a reusable function or sklearn‐compatible estimator; this makes CV and grid‐search much simpler.
2. **Logging & config files**
   * Instead of hard‐coded paths and constants, define all parameters (window lengths, offsets, α values) in a JSON or YAML config; log every run’s config and random seeds for reproducibility.
3. **Parallelization & efficiency**
   * You’re looping channels in Python; consider vectorizing or using joblib’s Parallel to speed up large subject/channel sets.

**In summary**

Your current script is a solid starting point, but to transform it into a rigorous iEEG decoding pipeline you should:

* Adopt **nested cross‐validation** or hold-out splits for encoding
* Baseline-normalize every trial and systematically reject artifacts
* Move beyond raw accuracy: report per‐class metrics, effect sizes, permutation p-values and correct for multiple comparisons
* Aggregate at the ROI/subject level with mixed-effects models
* Explore **temporal generalization** and simpler feature‐based baselines
* Encapsulate in a fully reproducible, configurable pipeline

Implementing these changes will give you statistically sound, neurophysiologically interpretable results—and guard against over-fitting and data leakage.

**1) Deep-dive list of possible script improvements for parallelized Minirocket model (binary class one vs all for each channel).**

1. **n\_cores control**  
   Always honour your chosen number of workers instead of hard-coding 32 or 64.
2. **Kernel caching**  
   MiniRocket(random\_state=42) creates the same 10 000 kernels each time; by instantiating one outside the per-class loop you avoid that overhead 4× per channel.
3. **Error isolation**  
   Wrapping the body of process\_channel in try/except ensures a single channel’s unexpected edge case doesn’t kill the entire job.
4. **Reproducibility**  
   Setting np.random.seed(42) at the top gives reproducible undersampling across runs.
5. **Logging vs print**  
   For production, you’d swap print() for the logging module so you can capture levels and timestamps.
6. **Class-weight instead of down-sampling**  
   Many sklearn classifiers support a class\_weight="balanced" flag; you could switch to a logistic or SVM with balanced weights to avoid throwing away data.
7. **Progress bars**  
   Tools like tqdm wrapped around your Parallel iterator can give live feedback on progress.
8. **GPU acceleration**  
   A PyTorch‐based Rocket implementation (e.g. rocket-learn) could push transforms onto your L90 GPU.
9. **Batching**  
   Instead of one channel per task, grouping channels (or even subjects) per worker can reduce scheduling overhead.

**Recommended next steps:**

* Run your permutation-p tests as you’ve coded.
* Then correct those \*\_p\_perm values across channels/classes via FDR or, better, a cluster-based permutation that groups adjacent channels or time bins (Maris & Oostenveld, 2007).
* Use the binomial p’s as a sanity check—if both agree you have an extra confidence boost.

**References**

* Maris, E., & Oostenveld, R. (2007). Nonparametric statistical testing of EEG‐ and MEG‐data. *Journal of Neuroscience Methods*.
* Stelzer, J., Chen, Y., & Turner, R. (2013). Statistical inference and multiple testing correction in classification-based multi-voxel pattern analysis (MVPA): random permutations and cluster size control. *NeuroImage*.
* Combrisson, E., & Jerbi, K. (2015). Exceeding chance level by chance: The caveat of theoretical chance levels in brain signal classification and statistical assessment of decoding accuracy. *Journal of Neuroscience Methods*.