Report: RT Regression Model Selection (Ridge Pooling vs Hierarchical Ridge vs Lasso Baselines)

Bioinformatics Team

December 25, 2025

1 Executive summary

We evaluate model **form** (not feature engineering) for RT regression, focusing on:

- **Accuracy**: global RMSE/MAE on realtest.
- **Uncertainty quality**: empirical coverage of nominal 95% prediction intervals and their average width.
- **Practicality**: training time and ability to support cold-start predictions.

We compare two ridge-style model forms (and two lasso baselines) for lib208 and lib209 under **cap100 training** and **realtest evaluation**:

- **Ridge (supercategory, PyMC)**: a fast collapsed ridge model fit per `species_cluster`.
- **Ridge (supercategory, sklearn)**: a per-group ridge baseline that writes coefficient summaries and an approximate predictive distribution.
- **Ridge (partial pooling)**: a hierarchical ridge model fit per `species` with priors pooled through `species_cluster`.
- **Lasso (supercategory)** and **Lasso (species)** baselines from Sally's legacy model bundles.

Key result: **Ridge (partial pooling)** improves RMSE and yields much tighter prediction intervals with coverage close to nominal (0.95), but costs **hours of training time** (ADVI). **Ridge (supercategory, PyMC)** is essentially instantaneous and safe, but is **over-conservative** (coverage $\sim$0.99) and produces much wider windows. **Ridge (supercategory, sklearn)** matches the same RMSE (ridge-equivalent point predictions) but produces **much narrower windows that under-cover** (coverage $\sim$0.92–0.94), which would systematically exclude true peaks in downstream peak assignment.

Recommendation: Use **Ridge (partial pooling)** as the primary model form when we can afford offline training (e.g., periodic retrains), and keep **Ridge (supercategory, PyMC)** as a fast fallback / debugging baseline.

2 Introduction

Downstream peak assignment benefits from RT predictions that are both accurate and accompanied by well-calibrated uncertainty (for candidate windowing). At the same time, we require a model form that:

- generalizes to **new species not present in training** (cold-start), and
- remains stable in the **long tail** of sparse compound/group history.

This report summarizes our current cap100 $\rightarrow$ realtest experiments, compares ridge pooling variants to two lasso baselines, and provides a recommendation on model form.

3 Methods

3.1 Data and group definitions

The production RT CSVs contain rows of curated RT observations with run-level covariates (`IS*`/`RS*`/`ES_` $\times$), and identifiers including:

- `species_cluster`: supercategory identifier (matrix/supercategory),
- `species`: subgroup identifier nested under `species_cluster`,
- `comp_id`: compound id.

The stage-1 artifact stores coefficient summaries per group

$$g = (\texttt{group\_id}, \texttt{comp\_id}),$$

where `group_id` is derived from a configured `group_col`. In this work:

- `group_col=species_cluster` (supercategory ridge),
- `group_col=species` (hierarchical ridge; nested within supercategory).

3.2 Feature set

Let $\boldsymbol{x}_i$ be the vector of run-level covariates for test row i. For pooling comparisons in this report we use **linear anchors only** (no poly2) to avoid confounding "pooling effect" with "feature expansion effect".

3.3 Models

All models are linear-in-features at prediction time and differ in grouping and pooling structure.

3.3.1 Ridge (supercategory, PyMC): collapsed per-`species_cluster` ridge

For each group g (here: (`species_cluster`, `comp_id`)) we model RT:

$$y_{gi} = b_g + \boldsymbol{x}_{gi}^\top \boldsymbol{w}_g + \epsilon_{gi}, \quad \epsilon_{gi} \sim \mathcal{N}(0, \sigma_g^2).$$

With a Gaussian ridge prior $\boldsymbol{w}_g \sim \mathcal{N}(\mathbf{0}, \tau_w^2 \boldsymbol{I})$, the posterior mean matches standard ridge regression and the posterior covariance is available in closed form. This model trains extremely quickly.

3.3.2 Ridge (supercategory, sklearn): per-group ridge coefficient summaries

This baseline trains an independent ridge regression for each group $g = (\texttt{species\_cluster}, \texttt{comp\_id})$ using the same linear run covariates. It writes a production-friendly artifact of per-group coefficient posterior summaries (`beta_hat`, `beta_cov`, `sigma2_mean`) and uses a Normal approximation to produce per-row prediction intervals. When we match the ridge penalty λ to the PyMC collapsed ridge and use the same evaluation code, the *point predictions* match up to numerical precision, but the *interval widths and coverage* can differ substantially.

3.3.3 Ridge (partial pooling): hierarchical priors for `species` within `species_cluster`

The partial pooling model keeps the subgroup grouping (`species`) but introduces hierarchical pooling for:

- **intercepts** with compound-aware structure, and
- **slope prior means** ("slope heads") with supercategory shrinkage.

Intercept prior (`intercept_prior=comp_hier_supercat`). For each subgroup c with parent supercategory $s(c)$ and each compound k:

$$b_{(c,k)} \sim \mathcal{N}\left(t_0 + \mu_c + \alpha_k, \tau_b\right),$$

with

$$\mu_c \sim \mathcal{N}\left(\mu_{s(c)}, \tau_{\mu,s(c)}\right), \quad \mu_s \sim \mathcal{N}(0, \tau_{\mu,\text{supercat}}), \quad \alpha_k \sim \mathcal{N}(0, \tau_{\text{comp}}).$$

Slope head (`slope_head_mode=cluster_supercat`). We learn a slope mean for each subgroup $\boldsymbol{m}_c \in \mathbb{R}^p$ and use it as the prior mean for group slopes $\boldsymbol{w}_{(c,k)}$. We pool slope heads through the supercategory:

$$\boldsymbol{m}_c \sim \mathcal{N}\left(\boldsymbol{m}_{s(c)}, \tau_{w,s(c)}\right), \quad \boldsymbol{m}_s \sim \mathcal{N}\left(\boldsymbol{m}_0, \tau_{w,\text{supercat}}\right).$$

Inference. We use ADVI for scalability. The exported coefficient summaries incorporate learned prior means so pooling shifts mean predictions as intended.

3.3.4 Lasso baselines

We include two legacy eslasso baselines for context:

- **Lasso (supercategory)**: one lasso per `species_cluster`.
- **Lasso (species)**: a smaller set of per-matrix models ("local models") selected from a `species_raw` mapping; this baseline does **not** score all rows.

Lasso "intervals" are window-based (`pred` $\pm$ `window`) rather than derived from a probabilistic predictive distribution. We still report their empirical "coverage95" for comparison, but it should not be interpreted as nominal calibration in the same way as ridge models.

3.4 Evaluation metrics

Let y_i be observed RT and $\hat{y}_i$ be the point prediction on realtest (units: minutes).

- **RMSE**: $\sqrt{\frac{1}{n}\sum_i (\hat{y}_i - y_i)^2}$.
- **MAE**: $\frac{1}{n}\sum_i |\hat{y}_i - y_i|$.
- **Coverage@95%** (ridge models): with predicted standard deviation $\hat{\sigma}_i = \sqrt{\sigma_g^2 + \boldsymbol{x}_i^\top \widehat{\mathrm{Cov}}(\boldsymbol{\beta}_g)\boldsymbol{x}_i}$, we define the nominal 95% interval as $\hat{y}_i \pm 1.96\,\hat{\sigma}_i$ and compute the fraction of test points inside.
- **Mean interval width** (ridge models): $\frac{1}{n}\sum_i 2 \cdot 1.96 \cdot \hat{\sigma}_i$.
- **Coverage@95% / width (lasso)**: computed using the legacy window: $|\hat{y}_i - y_i| \le w_i$ with width $2w_i$.

We stratify by **training support bin** of each (`group_id`, `comp_id`) group using bins:

$$\{\le 1,\ 2,\ 3\text{–}5,\ 6\text{–}10,\ 11\text{–}20,\ 21\text{–}50,\ 51\text{–}100,\ >100\}.$$

3.5 Reproducibility

Ridge experiments were generated by:

```
./scripts/run_rt_prod.sh --cap 100 --libs 208,209
./scripts/run_rt_prod_eval.sh --cap 100 --libs 208,209
```

and plots/tables in this report were regenerated by:

```
./scripts/plot_rt_multilevel.sh --cap 100 --libs 208,209
```

using the run directory pointer in `output/rt_prod_latest.txt`.

4 Results

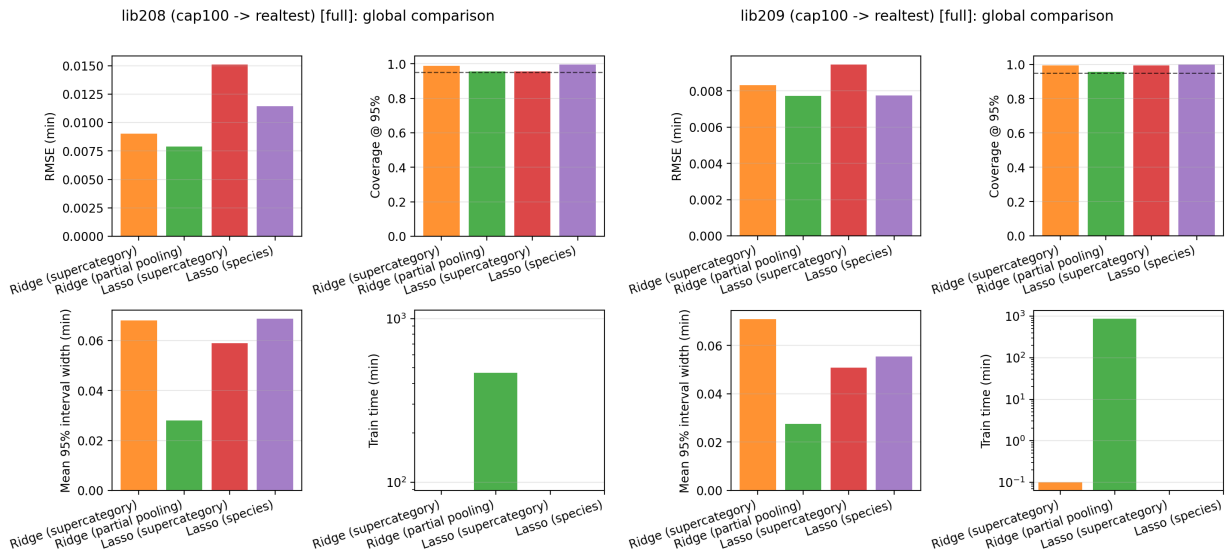
4.1 Global performance

Table 1 summarizes global metrics on realtest for lib208 and lib209. Lasso baselines do not score all rows; **Rows scored** reflects applicability.

Table 1: Global realtest metrics for cap100-trained models (linear features).

Lib	Model	RMSE	MAE	Cov95	Width95	Rows scored	Train (min)
208	Ridge (supercategory, PyMC)	0.009023	0.004673	0.987	0.068109	100.0%	0.0
208	Ridge (supercategory, sklearn)	0.009023	0.004673	0.943	0.023534	100.0%	0.0
208	Ridge (partial pooling)	0.007877	0.003827	0.956	0.027978	100.0%	465.0
208	Lasso (supercategory)	0.015081	0.007956	0.954	0.058908	94.7%	–
208	Lasso (species)	0.011425	0.006510	0.994	0.068647	51.2%	–
209	Ridge (supercategory, PyMC)	0.008317	0.004677	0.993	0.070829	100.0%	0.1
209	Ridge (supercategory, sklearn)	0.008317	0.004677	0.920	0.021393	100.0%	0.0
209	Ridge (partial pooling)	0.007718	0.004298	0.956	0.027551	100.0%	860.9
209	Lasso (supercategory)	0.009439	0.004954	0.993	0.050886	97.8%	–
209	Lasso (species)	0.007741	0.004346	0.998	0.055400	82.7%	–

Interpretation. Relative to Ridge (supercategory, PyMC), Ridge (partial pooling) reduces RMSE by 0.00115 min (0.069 s) on lib208 and 0.00060 min (0.036 s) on lib209. It also shrinks average 95% interval width by about 0.040–0.043 min (2.4–2.6 s) and moves coverage closer to the nominal 0.95 target. The sklearn supercategory ridge matches the same RMSE (ridge-equivalent means) but substantially under-covers, which is undesirable when the prediction interval is used as an RT filter in peak assignment.



(a) lib208 (b) lib209

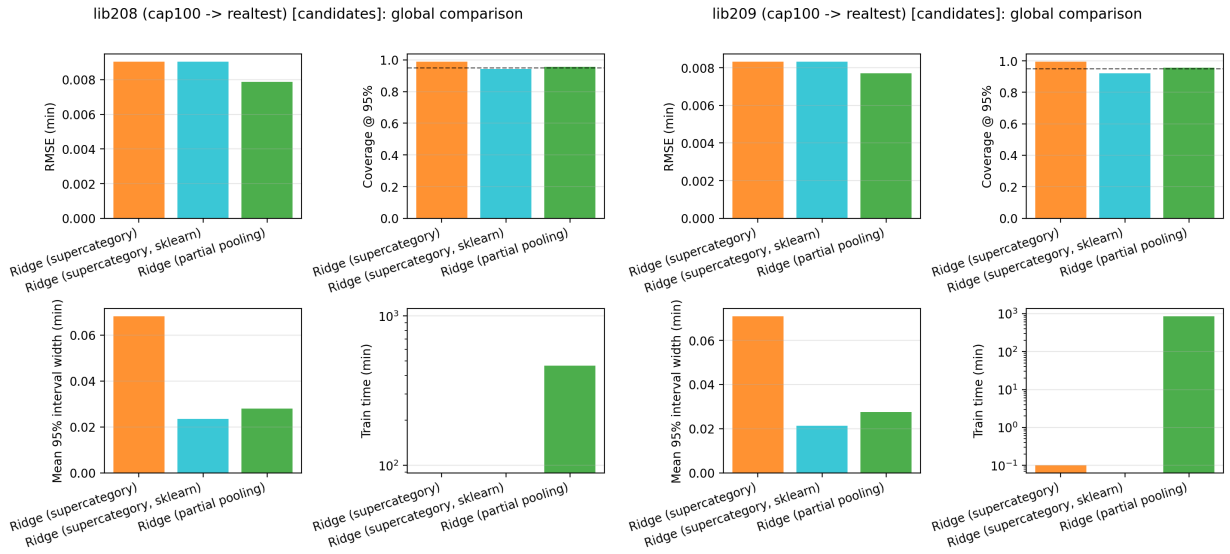
Figure 1: Global comparison across models (cap100 training, realtest evaluation). Panels show RMSE, coverage@95%, mean interval width, and training time.

4.2 Comparison vs sklearn ridge baseline

The sklearn baseline and the PyMC supercategory model are both ridge regression models over the same run covariates. When we fix the ridge penalty λ to the same value in both implementations, their posterior means (and therefore their point predictions) match up to numerical precision. This is expected: with a Gaussian ridge prior on slopes and a flat intercept prior, the posterior mean equals the ridge solution.

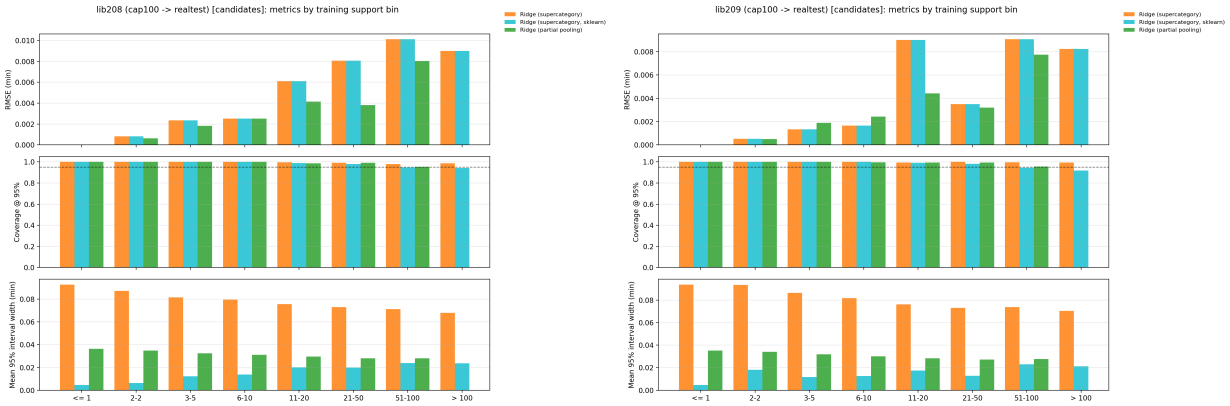
However, the **prediction intervals** differ because they depend on how we estimate (and propagate) predictive variance. In particular, the sklearn coefficient-summary artifact estimates a separate noise scale (`sigma2_mean`) for each (`species_cluster`, `comp_id`) group, which can lead to **overly optimistic** intervals in practice. Empirically this shows up as substantially **narrower windows** but **lower-than-nominal** coverage (0.92–0.94). In a peak-assignment setting where we use the interval as a hard RT gate, this undercoverage translates directly into a higher *miss rate*: the true peak is excluded from consideration more often.

Figure 2 shows the three candidate ridge variants (PyMC supercategory, sklearn supercategory, and PyMC partial pooling). Figure 3 shows the same comparison stratified by training support.



(a) lib208 (b) lib209

Figure 2: Global comparison of ridge candidates only. The sklearn supercategory model achieves narrower windows by under-covering (coverage below 0.95). The PyMC supercategory model over-covers with wider windows. PyMC partial pooling is closest to the desired operating point: narrow windows with near-nominal coverage.

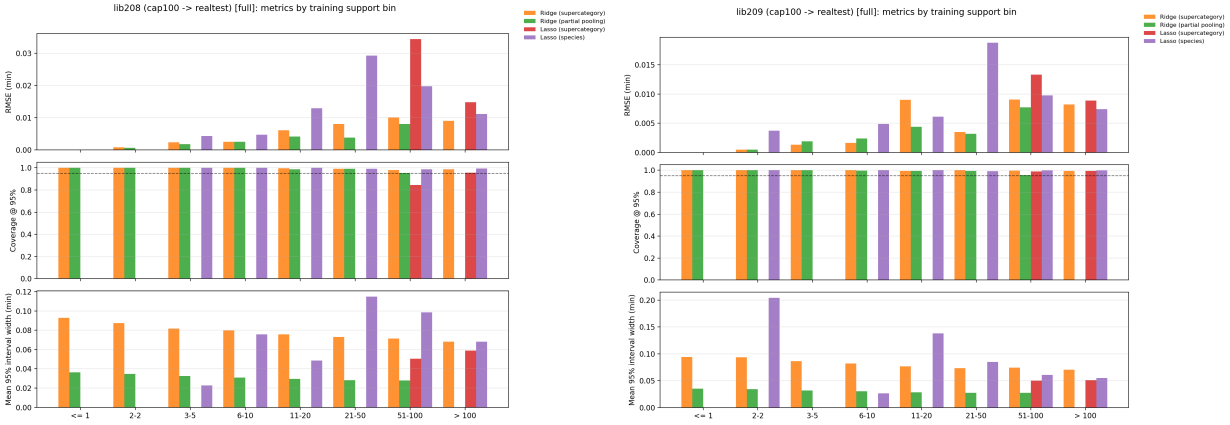


(a) lib208 (b) lib209

Figure 3: Ridge candidate comparison by training support bin. The same pattern holds across the long tail: sklearn tends to be narrower but under-calibrated; PyMC partial pooling tightens intervals while maintaining near-nominal coverage.

4.3 Performance by training support bin

Figure 4 breaks metrics down by training support bin (long tail to head).

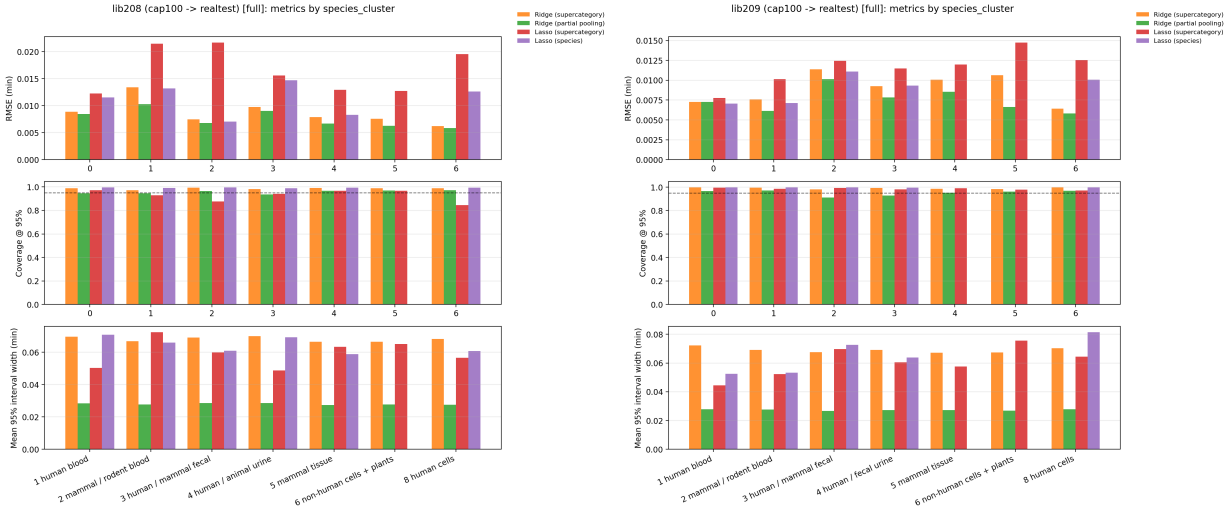


(a) lib208 (b) lib209

Figure 4: Metrics by training support bin for each model. Coverage@95% is empirical coverage of the nominal 95% prediction interval (ridge models) or window interval (lasso baselines).

4.4 Performance by species_cluster

Figure 5 aggregates metrics by `species_cluster`.



(a) lib208 (b) lib209

Figure 5: Metrics by `species_cluster` (supercategory). These aggregates reflect within-supercategory performance stability and highlight where models under/over-estimate uncertainty.

5 Discussion

Accuracy vs uncertainty for peak assignment. For downstream peak assignment, tighter uncertainty windows (with maintained calibration) directly reduce the set of candidate peaks per compound, improving

precision. Ridge (partial pooling) produces substantially narrower intervals while keeping coverage close to nominal; Ridge (supercategory, PyMC) is consistently over-conservative with much wider windows.

Why we do not prefer sklearn intervals (as-is). The sklearn ridge baseline produces very narrow intervals, but empirically under-covers the nominal 95% target. In a peak assignment pipeline that uses RT intervals as a hard filter, under-coverage becomes a direct failure mode: the true peak is excluded more often. The PyMC partial pooling model achieves a better operating point: narrow windows with coverage close to nominal, so it reduces candidate load without sacrificing recall.

Cost of hierarchy. The benefit of partial pooling comes at a large training cost: $\sim$7.8 hours (lib208) and $\sim$14.3 hours (lib209) in these experiments. If we retrain infrequently (e.g., periodic offline retrains), this may be acceptable; if we need rapid iteration, Ridge (supercategory, PyMC) remains valuable.

Cold-start behaviour. Both ridge forms can provide predictions for new species as long as the row can be assigned to an existing `species_cluster`. The hierarchical model additionally defines a principled prior over unseen `species` within a supercategory, but production usage would require explicitly sampling or constructing coefficients for unseen groups (future work).

Lasso baselines. Lasso (supercategory) is generally weaker in RMSE than ridge candidates. Lasso (species) can look competitive on lib209 but scores only a subset of rows (and therefore requires a fallback model). Its interval is a fixed window, so coverage should be interpreted as "fraction inside the window" rather than probabilistic calibration.

6 Conclusion and recommendation

Based on current cap100 $\rightarrow$ realtest results, **Ridge (partial pooling)** is the best overall model form for peak assignment: it improves RMSE and substantially tightens predictive windows with near-nominal coverage. The main downside is training time (hours).

We recommend:

- **Primary**: Ridge (partial pooling) for production-quality coefficients and calibrated intervals.
- **Fallback / fast iteration**: Ridge (supercategory, PyMC) as a fast, stable baseline and safety net.
- **sklearn ridge**: useful as a fast baseline and for debugging; avoid using its intervals for gating unless we explicitly calibrate them to hit nominal coverage.

7 Future extensions

- **Peak-assignment integration:** use the predictive distribution as a likelihood term (not only a hard gate) and tune operating points explicitly in terms of candidate count vs miss rate.

- **Calibration as a first-class step:** fit a small global (or per-supercategory) predictive-scale calibration on a held-out validation set to hit nominal coverage while minimizing window width.
- **Active learning loop closure:** choose which species/compounds to curate next based on assignment ambiguity (high candidate count), long-tail support, and calibration failures; retrain periodically and measure end-to-end impact.
- **Instrument transfer:** add an explicit transfer layer (e.g., per-instrument offset/scale, or hierarchical run-drift components) so new instruments can adapt using limited anchor data without losing calibration.
- **Cold-start matrices and unseen chemicals:** extend intercept priors to chemistry-informed backoff (e.g., embedding or class-conditioned priors) so we can produce sensible means and honest uncertainty for unseen compounds and/or new matrices.
- **Training speed and operational simplicity:** reduce ADVI wall time (structured VI, fewer parameters, better initialization) and provide clean production scripts with sensible hardcoded defaults for reproducibility.