Compressed Computation is (probably) not Computation in Superposition

Jai Bhagat[*]
Enigma

Sara Molas-Medina[*]
Independent

Giorgi Giglemiani
UK AI Security Institute

Stefan Heimersheim[†]
FAR.AI

Abstract

We study whether the Compressed Computation (CC) toy model Braun et al. [2025] is an instance of computation in superposition. The CC model appears to compute 100 ReLU functions with just 50 neurons, achieving a better loss than expected from only representing 50 ReLU functions. We show that the model mixes inputs via its noisy residual stream, corresponding to an unintended mixing matrix in the labels. Splitting the training objective into the ReLU term and the mixing term, we find that performance gains scale with the magnitude of the mixing matrix and vanish when the matrix is removed. The learned neuron directions concentrate in the subspace associated with the top 50 eigenvalues of the mixing matrix, suggesting that the mixing term governs the solution. Finally, a semi-non-negative matrix factorization (SNMF) baseline derived solely from the mixing matrix reproduces the qualitative loss profile and improves on prior baselines, though it does not match the trained model. These results suggest CC is not a suitable toy model of computation in superposition.

1 Introduction

Superposition lets networks represent many sparse features in fewer dimensions [Elhage et al., 2022, Gurnee et al., 2023, Bricken et al., 2023]. The related question, *computation in superposition*, asks whether models can implement more nonlinear functions than they have nonlinearities when inputs are sparse [Hänni et al., 2024, Bushnaq and Mendel, 2024, Adler and Shavit, 2024, Olah et al., 2025]. Braun et al. [2025] introduce a toy model of Compressed Computation (CC) which seemingly implements computation in superposition: it appears to compute 100 ReLU functions of sparse inputs using only 50 neurons.

We re-examine the CC model, and find that its performance—a better loss than a naive baseline computing only 50 ReLU functions—is likely due to noisy labels introduced by Braun et al. [2025]'s architecture. Our key results are:

- The CC model's performance advantage is dependent on residual stream noise that mixes different inputs into the labels (in addition to the ReLU target). The model does not beat baselines without this effective "mixing matrix".

- The CC model's performance scales smoothly with the mixing matrix's magnitude (higher is better, up to a limit). We also find that the trained model focuses on the top 50 singular vectors of the mixing matrix (all its neuron directions mostly fall into this subspace), mostly ignoring the 50 others.

---

[*]These authors contributed equally to this work and are listed in alphabetical order.
[†]Correspondence to `stefan@far.ai`. Work done while at Apollo Research.

- We introduce a new baseline model derived from the SNMF of the mixing matrix alone which beats the previous naive baseline, and qualitatively matches the CC model's loss profile. However, the SNMF weights qualitatively differ from the trained model's weights, and we don't fully recover the trained model's loss.

2 Methods

We show the residual CC architecture with fixed random embedding W_E [Braun et al., 2025] is equivalent to a 1-layer MLP trained on $y = \text{ReLU}(x) + Mx$ with $M = I - W_E W_E^T$ (details in appendix A.1). Inputs $x_{0..99}$ are nonzero with probability p and drawn uniformly from $[-1, 1]$, otherwise zero. We vary M across three conditions: $M = 0$ (clean), random $M \sim \mathcal{N}(0, \sigma^2)$, and the embedding-induced $M = I - W_E W_E^T$. Only W_{in} and W_{out} are trained. Training details are provided in appendix B. We provide our core code and scripts to reproduce all figures at this URL.
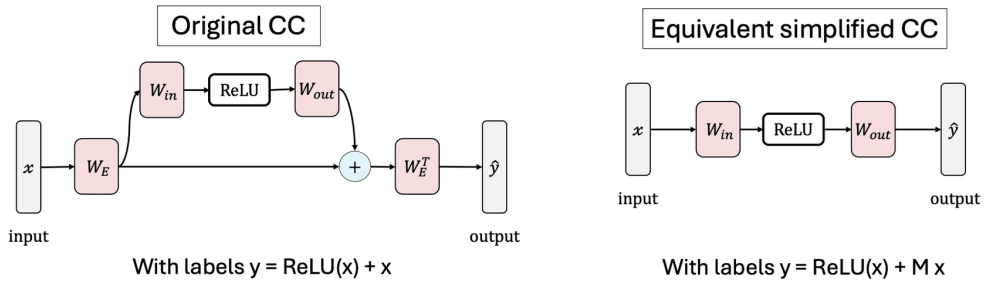

Figure 1: The original model architecture from Braun et al. [2025], and our simpler equivalent model. The labels for our (new) model are $y_i = \text{ReLU}(x_i) + \sum_j M_{ij} x_j$. The matrix M mixes other inputs x_j into the label y_i. Thus the MLP needs to learn both the ReLU term, and the mixing term.

3 Results

Simplified model reproduces CC results.   We reproduce the results of Braun et al. [2025] with our simplified model. Our 1-layer MLP model is trained on $y = \text{ReLU}(x) + Mx$ with a mixing matrix $M = 1 - W_E W_E^T$. This mixing matrix accounts for the noisy residual stream contribution in Braun et al. [2025]'s model. Our model produces the same results, and we reproduce their figures in appendix C. Figure 2 (left) shows that the "better-than-naive[1] at low p / worse at high p" profile is reproduced by the embedding-like M, as well as by a random $M \sim \mathcal{N}(0, 0.02)$. We find that a symmetric random M matrix matches Braun et al. [2025]'s results almost exactly.

CC loss advantage requires a nonzero mixing matrix.   Without the noise-induced mixing matrix, the model does not beat the naive loss (yellow line in Figure 2 left) for sparse inputs.[2] If the model was achieving a good loss due to computation in superposition, we would expect the loss to still beat the naive loss even with $M = 0$. Furthermore, we confirm that a model trained on $M \neq 0$ immediately reverts to the naive loss when fine-tuned on an $M = 0$ dataset (Figure 3).

CC loss advantage scales with the magnitude of M.   We train CC models over a range of noise scales $M \sim \mathcal{N}(0, \sigma)$ with σ from 0 to 0.08, as shown in Figure 2 (right). We find that the loss decreases with σ (for $\sigma \lesssim 0.03$). This direct correspondence strongly suggests that the loss advantage is coming directly from the noise term, rather than from computing the 100 ReLU target functions (computation in superposition).

---

[1]The naive baseline of representing 50 ReLU functions perfectly achieves a loss of 0.0833 on a clean dataset (details in appendix A.2).

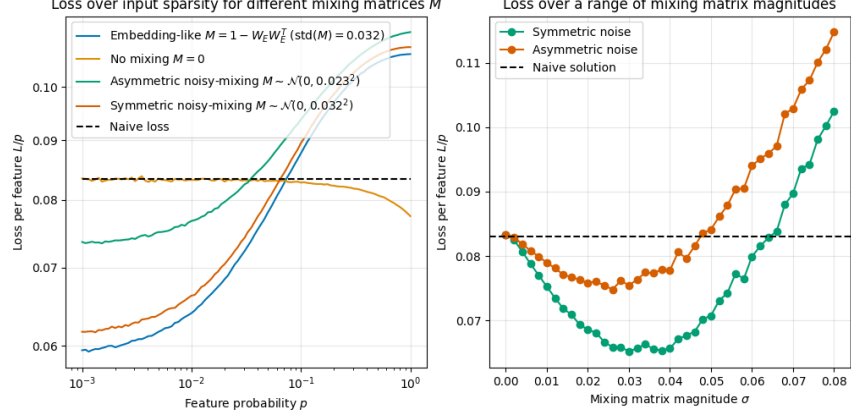[2]The dense regime is a different mechanism, discussed in appendix D.

Figure 2: Left: With $M \neq 0$ (embed-like, random, or symmetric), the model beats the naive loss at high sparsity. Labels with embed-like M allow for the lowest loss, but random M matrices behave similarly. Right: Optimal loss decreases with noise scale σ for small σ and increases again at large σ.
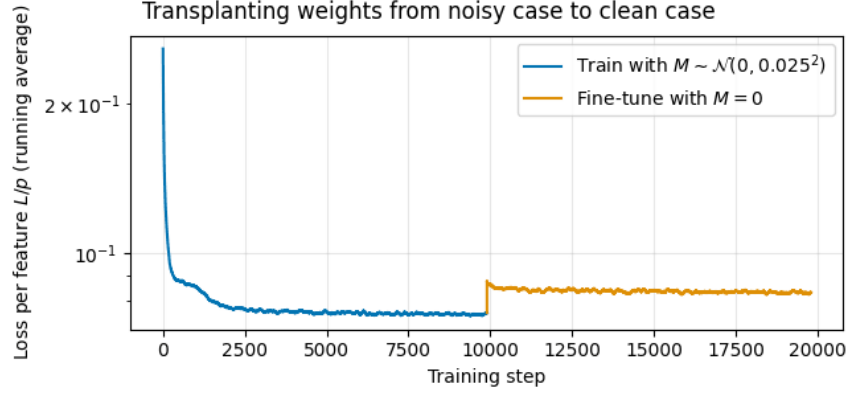

Figure 3: Transplant to clean labels. A model trained with $M \neq 0$ immediately reverts to the naive loss when fine-tuned on $M = 0$, indicating the lower loss is not just due to training dynamics.

Potential mechanism behind the Compressed Computation model. We notice that the direction read (W_{in}) and written (W_{out}) by each MLP neuron lies mostly in the 50-dimensional subspace spanned by the positive eigenvalues of M (Figure 4 left, top panels) or the top 50 singular vectors of $I + M$ (bottom panels). We confirm this by measuring the cosine similarity of singular- and eigenvectors with their respective projections through ($W_{\text{out}}W_{\text{in}}$). We find that only the top ~ 50 vectors are represented fully, with a sharp drop-off in similarity around index 50 (Figure 4 right).

We further analyze the trained model's weights and find (i) the entries of the M matrix are strongly correlated with the entries of $W_{\text{out}}W_{\text{in}}$ (Figure 5 left), and (ii) W_{in} is mostly non-negative and both weight matrices are sparse (Figure 5 right). Given this we attempt to design hand-coded model weights using the SNMF of M, setting W_{in} to the non-negative factor. We find that this SNMF solution beats the naive loss for a range of noise scales σ (Figure 6), in a qualitatively similar pattern to the trained model (Figure 2 right). However, the SNMF solution does not reach a loss as low as the trained model, and its weights are less sparse than the trained weights (Figure 5 right).

4 Conclusion

Our work sheds light on the mechanism behind the Braun et al. [2025] model of compressed computation. We ruled out the hypothesis that the embedding noise was only required for the training of the model, and instead showed that the noise-induced mixing matrix is what allows the model to beat the naive baseline. We provided an improved SNMF baseline that explains some but not all of
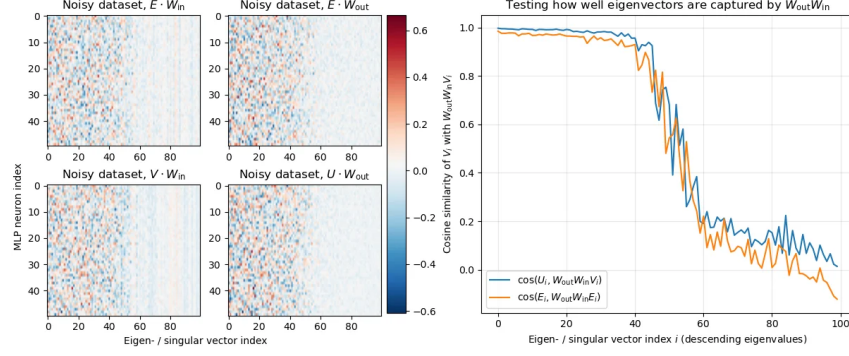
Figure 4: Left: Neuron read/write directions align with the top ~ 50 eigen/singular vectors of M. Right: The linear map $W_{\text{out}}W_{\text{in}}$ preserves those top directions while strongly attenuating the rest.
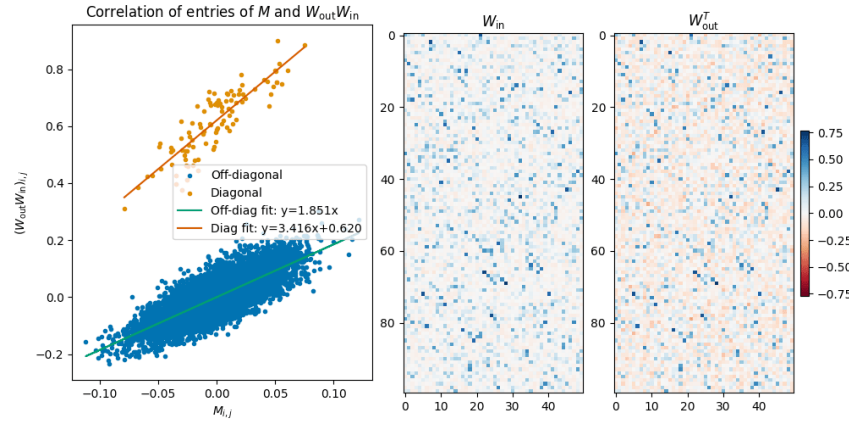

Figure 5: Left: Entries of the M matrix strongly correlated with entries of $W_{\text{out}}W_{\text{in}}$ (with an offset for the diagonal entries). Right: W_{in} is mostly nonnegative and both weight matrices are sparse.

the trained model's performance and properties. While we were not able to fully reverse-engineer the Compressed Computation model, we hope that our work inspires new attempts to find a toy model of computation in superposition, and further reverse-engineer the CC model.

References

Micah Adler and Nir Shavit. On the complexity of neural computation in superposition, 2024.

Dan Braun, Lucius Bushnaq, Stefan Heimersheim, Jake Mendel, and Lee Sharkey. Interpretability in parameter space: Minimizing mechanistic description length with attribution-based parameter decomposition, 2025.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E. Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. URL https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Lucius Bushnaq and Jake Mendel. Circuits in superposition: Compressing many small neural networks into one. Alignment Forum, 2024. URL https://www.alignmentforum.org/posts/roE7SHjFWEoMcGZKd/circuits-in-superposition-compressing-many-small-neural.
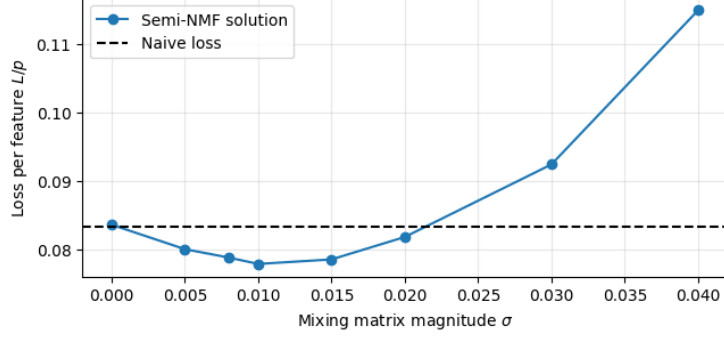
Figure 6: The semi-NMF baseline derived from M reproduces the qualitative loss-vs-σ curve, but does not reach as low a loss as the trained model.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022. URL `https://transformer-circuits.pub/2022/toy_model/index.html`.

Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing, 2023.

Kaarel Hänni, Jake Mendel, Dmitry Vaintrob, and Lawrence Chan. Mathematical models of computation in superposition, 2024.

Chris Olah, Nicholas L. Turner, and Tom Conerly. A toy model of interference weights. *Transformer Circuits Thread*, July 2025. URL `https://transformer-circuits.pub/2025/interference-weights/index.html`.

A Mathematical details

A.1 Equivalence of the CC model and 1-layer MLP

The CC residual architecture corresponds to the function

$$\hat{y}' = W_E^T \left(W_E x + W_{\text{out}}' \text{ReLU}(W_{\text{in}}' W_E x) \right) \tag{1}$$

$$= \underbrace{W_E^T W_E}_{I-M} \, x + \underbrace{W_E^T W_{\text{out}}' \text{ReLU}(W_{\text{in}}' W_E x)}_{W_{\text{out}} \text{ReLU}(W_{\text{in}} x)} \tag{2}$$

and is trained on the objective $y' = \text{ReLU}(x) + x$. This is equivalent to training a 1-layer MLP

$$\hat{y} = W_{\text{out}} \text{ReLU}(W_{\text{in}} x) \tag{3}$$

on the objective $y = \text{ReLU}(x) + Mx$.

A.2 Naive baseline loss

A noise-free model could implement 50 ReLU target functions perfectly, and ignore the other 50 target functions. The expected loss for an unrepresented feature is then $0.5 \int_{-1}^{1} \text{ReLU}(x)^2 \mathrm{d}x \approx 0.1667$. Thus the average loss per feature is 0.0833 for the naive baseline.

This only holds on the clean dataset ($M = 0$). With noisy labels, the naive solution would need to be adjusted to account for the noise. We do not investigate this further, and only indicate the 0.0833 line for comparison with Braun et al. [2025].

B   Training details

We use a batch size of 2048 and a learning rate of 0.003 with cosine scheduler. To control for training exposure we train all models for 10,000 non-empty batches. The only trainable parameters are W_{in} and W_{out} (not M). In some experiments we optimize a scale factor σ for the mixing matrix M simultaneously with the weights for convenience.

C   Analysis of the CC model

We reproduce the CC model of Braun et al. [2025] with our simplified architecture with $M = 1 - W_E W_E^T$ for a randomly generated $\mathbb{R}^{1000 \times 100}$ embedding matrix W_E, as well as for a random matrix $M \sim \mathcal{N}(0, 0.02)$.

In Figure 7 we show the loss per feature (that is, MSE loss divided by feature probability p) as a function of training and evaluation sparsity.
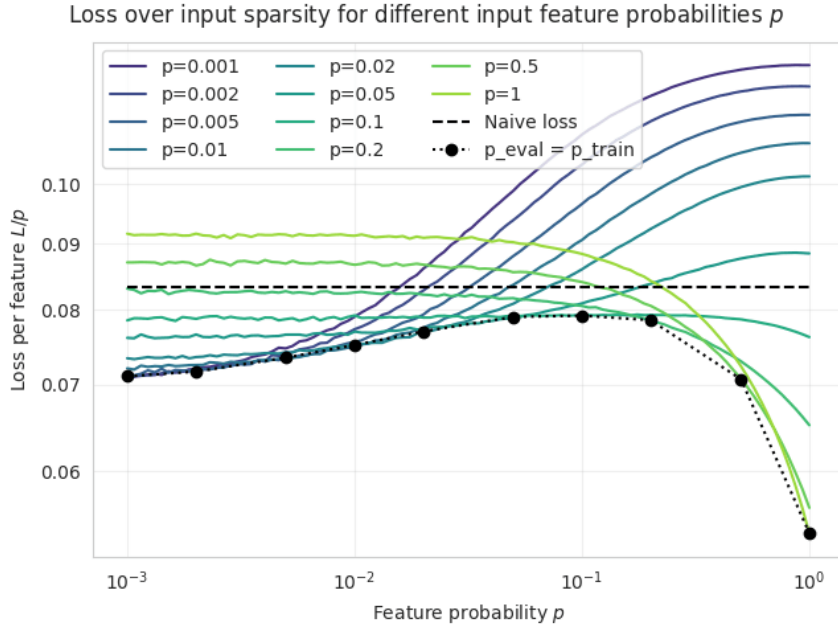

Figure 7: Loss per feature (L/p) as a function of evaluation sparsity. Each solid line corresponds to a model trained at a given sparsity. The models learn one of two solution types, depending on the input sparsity used during training: the CC solution (violet) or a dense solution (green). Both types beat the naive baseline (dashed line) in their respective regime. Black circles connected by a dotted line represent the results seen by Braun et al. [2025], where models were evaluated only at their training sparsity.

In the sparse regime (low probability $p \lesssim 0.05$) we find solutions that perform well on sparse inputs, and less well on dense inputs. They typically exhibit a similar input-output response for all features (Figure 8 left), and weights distributed across all features (Figure 9 left, equivalent to Figure 6 in Braun et al. [2025]). The maximally sparse case (exactly one input active) behaves very similar to $p \leq 0.01$.

In the dense regime (high probability $p \gtrsim 0.2$) we find solutions with a constant per-feature loss on sparse inputs, but a better performance on dense inputs. These solutions tend to implement half the input features with a single neuron each, while ignoring the other half (Figures 8 right and 9 right).
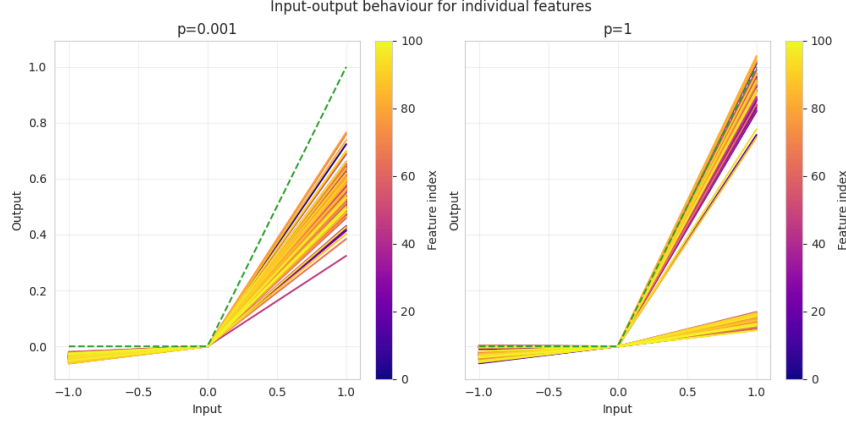
Figure 8: Input/output behaviour of the two model types (for one-hot inputs): In the CC solution (left panel), all features (inputs) are similarly-well represented: each input activates the corresponding output feature. In contrast, the dense solution (right panel) shows a strong (and more accurate) response for half the features, while barely responding to the other half. The green dashed line indicates the expected response under perfect performance.
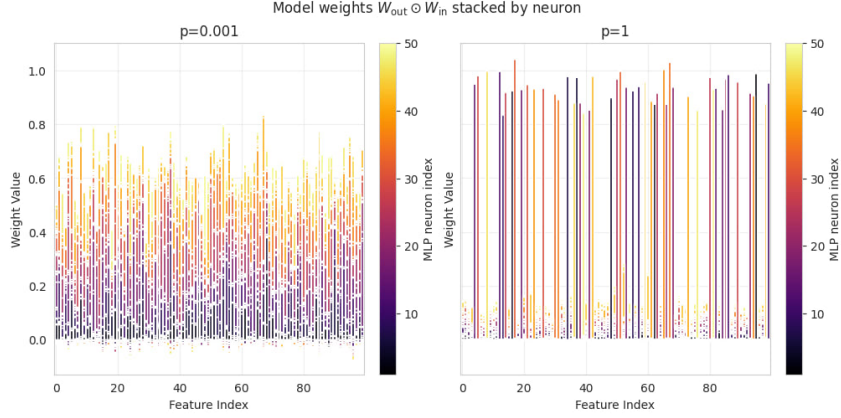

Figure 9: Weights representing each input feature, split by neuron. Each bar corresponds to a feature (x-axis) and shows the adjusted weight value from $W_{\text{out}} \odot W_{\text{in}}$, split by neuron index (color). The CC solution (left) combinations of neurons to represent each feature (to around 70%), whereas the dense solution (right) allocates a single neuron to fully ($\sim$100%) represent 50 out of 100 features.

D Mechanism of the dense solution

The dense regime is less interesting as it is not related to computation in superposition, however Figure 7 shows that the model can beat the naive baseline in the dense regime. In this section we provide an explanation of this behaviour, and a hand-coded model that matches the trained model's performance in the dense regime.

Revisiting Figures 8 and 9, we notice an intriguing pattern: roughly half of the features are well-learned, while the other half are only weakly represented. Our hypothesis is that the model represents half the features correctly, and approximates the other half by emulating a bias term. Our architecture does not include biases, but we think the model can create an offset in the outputs by setting the corresponding output weight rows to positive values, averaging over all features. This essentially uses the other features to, on average, create an offset.

We test this "naive + offset" solution on the clean dataset ($M = 0$) as the behaviour seems to be the same regardless of noise (but we did not explore this further), and find that the hardcoded naive + offset solution does in fact match the trained models' losses. Figure 10 left shows the optimal weight

7

value (offset strength; same value for every W_{out} entry of non-represented features), and Figure 10 right shows the corresponding loss as a function of feature probability. We find that the hardcoded models (dashed lines) closely match or exceed the trained models (solid lines). We thus conclude that the high density behaviour can be explained by a simple bias term, and not particularly interesting.
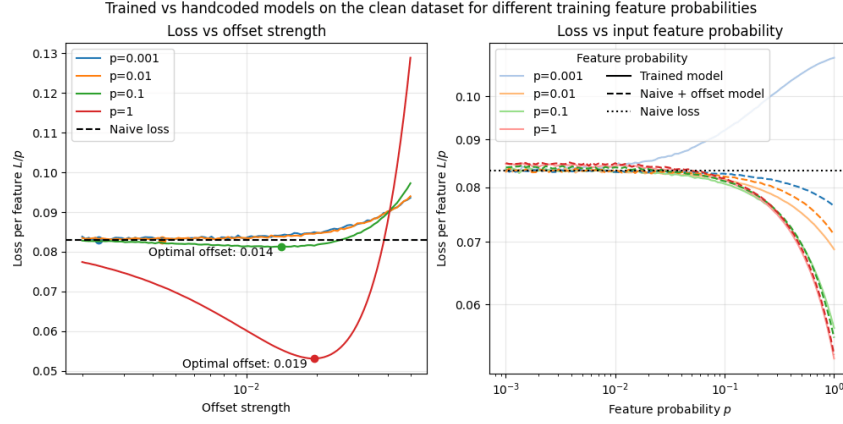

Figure 10: Left: A non-zero offset in the W_{out} entries of unrepresented features improves the loss in the dense regime. We determine the optimal value empirically for each input feature probability p. Right: This hand-coded naive + offset model (dashed lines) consistently matches or outperforms the model trained on clean labels (solid lines) in the dense regime. (Note that this plot only shows the clean dataset ($M = 0$) which is why no solution outperforms the naive loss in the sparse regime.)