

VRS: Void Rescue System for Regression in Transformers

Iker Moel Tacher
High School Student, Mexico

April 8, 2026

Abstract

We study regression-based token prediction in transformers, where the model predicts continuous embedding targets rather than vocabulary logits. Decoding therefore becomes geometric retrieval in embedding space, and a latent can contain useful token information while still decoding poorly if it falls in an ambiguous or weakly aligned region, which we interpret as a void. To address this failure mode, we introduce the Void Rescue System (VRS), a lightweight auxiliary decoder that maps raw regression latents to more token-aligned vectors before final decoding. Across corrected runs, VRS consistently improves both validation bits-per-byte (BPB) and nearest-neighbor top-1 accuracy (**nn_acc**) over raw decoding. In the three corrected 10-minute Parameter Golf runs with a 0.5M Rescuer and 17.58M total parameters, the rescued system reaches 1.8667 BPB on average and 50.51% peak **nn_acc**, with a stable BPB crossover at step 3600. Against three separately trained 10-minute regression-only baselines, which reach 2.0941–2.1301 BPB and 50.05–50.18% peak **nn_acc**, the short-run VRS system remains clearly better under the same budget. A 1M Rescuer reaches 1.8630 BPB and 50.62% peak **nn_acc**, crossing at step 3400, but exceeds the artifact limit. At larger scale, rescued decoding reaches 1.8480 BPB and 51.04% peak **nn_acc** in the 41-minute base run, 1.7570 BPB and 53.35% in the 2x one-hour run, and 1.7733 BPB and 55.34% in the 4x one-hour run; the 4x system still gains 0.6411 BPB over raw decoding and crosses stably by step 1600. These results are consistent with the hypothesis that regression latents can be informative before they are directly decodable, and that a very small decoder can improve decoding from such latents. Our aim is not to benchmark regression against standard classification models or argue that one paradigm is globally better; rather, we use regression as an experimental setting for studying transformer behavior and for testing whether a small decoder can correct regression-specific decoding errors.

1 Introduction

Language models are typically trained to predict the next token by direct classification over a vocabulary, and transformer architectures have become the standard backbone for that formulation [1]. Prior work has also argued that the output parameterization itself can act as a modeling bottleneck, making alternatives to the usual softmax head worth studying [2].

In this work, the model instead predicts a continuous target vector and is decoded through the shared embedding space, turning output prediction into geometric retrieval.

This regression setting introduces a failure mode: a predicted vector may contain useful token information yet still decode poorly because it falls in an ambiguous or weakly aligned region of

embedding space, which we interpret as a *void*. To address this problem, we introduce the *Void Rescue System* (VRS), a small auxiliary decoder that maps raw regression latents to more token-aligned representations before final decoding.

We use *void* throughout as a modeling hypothesis or geometric interpretation, not as a demonstrated mechanism. We also do not present VRS as a direct competitor to standard vocabulary-classification language models on absolute BPB or overall capability. The classification formulation appears here only as background for explaining the setup; our goal is not to decide whether regression or classification is better, but to use regression as an experimental lens for studying transformers and for testing whether a small auxiliary decoder can correct some of the decoding errors specific to regression-based transformers.

The main contributions of this draft are the following:

- a geometric interpretation of regression-decoding failure in terms of voids;
- VRS, a lightweight MLP rescue module applied to raw regression latents;
- empirical evidence that a 0.5M-parameter rescue model yields consistent BPB improvements across short, medium, and scaled runs.

2 Background and Motivation

2.1 Regression as an experimental setting

Standard language models predict the next token through vocabulary logits [1, 2]. In our setting, the model instead predicts continuous vectors that are decoded by retrieval against the shared embedding table. Both prediction branches are trained with a mean squared error (MSE) objective, and the embedding table serves both as the input lookup representation and as the target space for prediction, echoing prior work on tying or reusing input and output embeddings [3].

We mention the standard classification formulation only to situate the reader. The purpose of this paper is not to compare regression and classification head-to-head with benchmark metrics, but to experiment with regression as a way of probing transformer representations and decoding geometry.

This formulation is closer to representation prediction than to standard token classification, and it is conceptually closer to joint-embedding predictive approaches and recent language-oriented JEPA variants than to conventional next-token softmax training [4, 5]. It may likewise be useful for future multimodal systems that operate in a shared continuous space.

2.2 Voids as a decoding failure mode

We use *void* to denote an ambiguous or poorly aligned region of embedding space: a prediction that does not decode cleanly into a plausible token even though it may still contain useful token-level information.

Voids can arise either in low-density regions, where a prediction is weakly aligned with any token embedding, or between multiple competing token embeddings. In both cases, the failure is geometric rather than purely informational.

More broadly, continuous-output sequence models have been shown to depend strongly on target-space geometry, loss design, and decoding procedure, making geometric failure modes a reasonable object of study even if our specific “void” picture remains a modeling hypothesis [6–8].

2.3 Why a rescue model may help

If decoding fails because latents fall into voids, improving output quality may not require a larger main model. Instead, a small decoder can map v_{void} toward regions of embedding space that decode more reliably. Because it adds no new contextual information and is extremely small relative to the main architecture, it remains attractive under tight parameter and artifact constraints.

3 Problem Statement

Our system begins with a main predictor, which we refer to as the *Navigator* or Model A. Given an input context, Model A produces a continuous latent vector v_{void} , which may be informative yet not directly aligned with the token structure of the embedding space.

The raw decoding path uses this latent directly. Let $E \in \mathbb{R}^{V \times d}$ denote the shared trainable token embedding table, with vocabulary size V and latent dimension d . Raw vocabulary scores are obtained by projecting the latent against that table:

$$z_A = v_{\text{void}} E^\top.$$

For validation BPB, these scores are passed through the same output pipeline—projection, softcap, and softmax/cross-entropy against the target token—so that performance depends on the full predicted distribution rather than only the nearest token. By contrast, nearest-neighbor accuracy is computed from the top prediction under this same projection and therefore reflects top-1 recovery rather than full probabilistic calibration.

Although our setup does not retrieve from an external datastore, decoding through a learned space is related in spirit to prior nearest-neighbor retrieval methods in language modeling and machine translation [9, 10].

This motivates the core problem studied in this paper: a latent may already contain useful token information while still decoding poorly because it lies in a void or is otherwise misaligned for output scoring.

To address this, we introduce a secondary model, the Rescue model or Model B, which receives only v_{void} and outputs a transformed vector v_{rescued} :

$$v_{\text{rescued}} = f_{\text{VRS}}(v_{\text{void}}).$$

The problem is therefore one of recovery rather than generation.

4 Method

4.1 Base model: Navigator

The base predictor in our system is the Navigator (Model A), a causal decoder-only transformer trained in a regression setting rather than standard vocabulary classification. Instead of producing logits directly over the vocabulary, the Navigator outputs a continuous latent representation intended to match the embedding of the next token.

Our base configuration uses a vocabulary size of 1024, model dimension 512, 9 transformer layers, 8 attention heads, and 4 key-value heads with grouped-query attention (GQA). Each head has dimension 64, the MLP hidden size is 1024 (`mlp_mult` = 2), positional information is encoded with RoPE (`rope_base` = 10000), normalization is RMSNorm, and the attention operator is causal scaled dot-product attention. The main linear layers are bias-free. In larger experiments, we scale the number of layers to 18 in the 2x setting and 36 in the 4x setting while keeping the same latent dimension.

Internally, the stack is split into a first half that stores skip activations and a second half that consumes them, with `num_encoder_layers` = `num_layers` // 2.

Given an input token sequence, the Navigator looks up token embeddings, applies normalization, processes the sequence through the transformer stack, and applies a final normalization step. The resulting tensor is the latent prediction:

$$v_{\text{void}} \in \mathbb{R}^{B \times T \times 512}.$$

There is no final linear output head. For every token position in the sequence, the final normalized hidden state itself serves as the prediction, and training uses all positions rather than only the final token.

The Navigator is trained with an MSE objective against the embedding of the next token. The embedding table is learned as the model’s input lookup representation, but under this regression objective it acts as the target space for prediction rather than receiving direct gradient updates from output cross-entropy.

4.2 Void Rescue System

On top of the Navigator latent, we introduce the Void Rescue System (VRS), also referred to as the Rescue model or Model B, a lightweight auxiliary decoder applied before final decoding.

The Rescue model receives only v_{void} as input and produces a transformed vector v_{rescued} of the same dimension. It is a small, context-free MLP applied independently at every sequence position:

$$v_{\text{rescued}} = W_2 \text{SiLU}(W_1 v_{\text{void}}).$$

In the main competitive configuration, the layer dimensions are $512 \rightarrow 512 \rightarrow 512$, with no attention, no positional information, and no bias terms. This Rescue model contains 524,288 parameters.

We also evaluated a 1M-parameter Rescue variant in early short-run experiments, but its gains over the 0.5M version were negligible and it exceeded the 16 MB artifact limit. We therefore used the

0.5M Rescue model for the remainder of the experiments. Even in our largest run, that same model still improved validation BPB by 0.6411 in a system with approximately 67.2M total parameters.

Figure 1 summarizes the full training and inference pipeline, including the Navigator path, the Rescue model, and the shared embedding-based decoding interface.

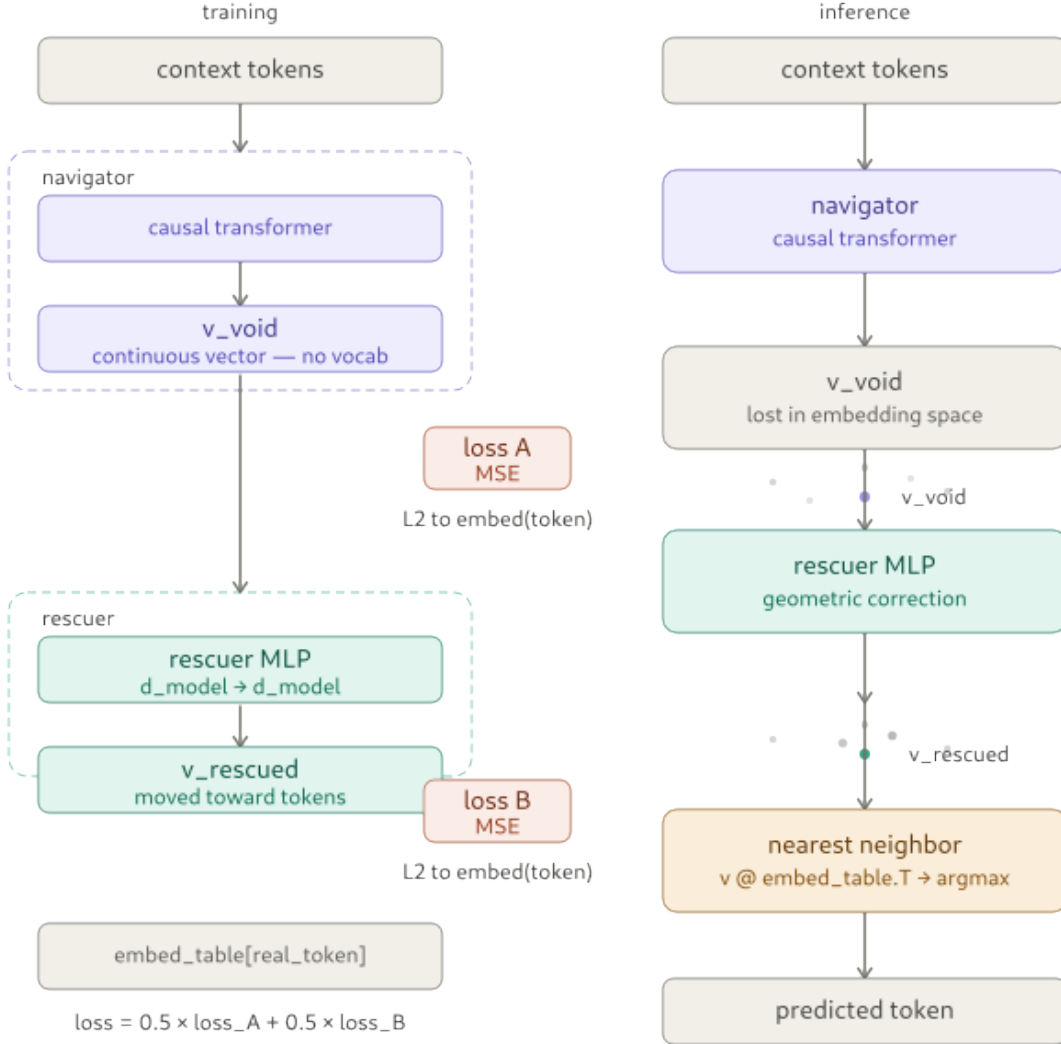


Figure 1: Architecture of the proposed system. Left: training-time pipeline with the Navigator, shared embedding targets, and the Rescue model losses. Right: inference-time decoding, where the rescuer maps v_{void} to a more token-aligned representation before final embedding-table decoding.

4.3 Decoding paths

We evaluate two decoding paths to separate raw latent quality from rescued latent quality.

Path A (raw decoding). In the first path, the Navigator output v_{void} is projected directly against

the shared trainable embedding table to produce vocabulary scores. This path measures how well the raw regression latent aligns with the token space without auxiliary correction.

Path B (rescued decoding). In the second path, v_{void} is first passed through VRS to produce v_{rescued} , which is then projected against the same embedding table to obtain vocabulary scores.

Both paths share the same decoding interface and embedding geometry; the only difference is whether the latent is decoded directly or first transformed by Model B. This isolates the effect of rescue from the contextual information already present in the Navigator latent. Because the reported VRS runs train Navigator and Rescuer jointly, Path A should be interpreted as an internal raw-decoding probe of the jointly trained system, not as a separately trained Navigator-only baseline.

For evaluation, projected scores are softly limited before computing the output distribution:

$$\text{softcap}_c(z) = c \tanh\left(\frac{z}{c}\right), \quad c = 30.$$

4.4 Metrics

We report multiple metrics to distinguish nearest-neighbor recovery from full distributional quality.

The metric `val_bpb` measures validation bits-per-byte for the full rescued system, after applying VRS and decoding from v_{rescued} . To compute this metric, the rescued latent is projected against the shared embedding table, passed through the softcap, and evaluated with softmax/cross-entropy against the target token. As a result, `val_bpb` depends on the full predicted distribution rather than only the top-ranked token.

The metric `val_bpb_A` is computed through the same projection and scoring pipeline, but using the raw Navigator output v_{void} without rescue. This metric probes the direct decodability of the raw Navigator latent within the jointly trained VRS system under the same shared embedding geometry. It should not be interpreted as a separately trained Navigator-only baseline.

The metric `nn_acc` measures nearest-neighbor top-1 accuracy, where “nearest” is defined by the maximum embedding dot-product score rather than by an explicit L^2 distance. Concretely, if logits are computed as $z = v_{\text{rescued}} E^\top$, then the predicted token is $\hat{y} = \arg \max_i z_i$, and `nn_acc` is the percentage of positions for which \hat{y} matches the target token. Unlike BPB, this metric does not capture full probabilistic calibration.

Thus, `nn_acc` and BPB need not move together.

5 Experimental Setting

5.1 Parameter Golf context

Our experiments were conducted in the context of the OpenAI Parameter Golf setting, which emphasizes strong performance under tight parameter, artifact-size, and compute constraints. This

makes efficiency a first-class concern: improvements must be judged not only by raw quality, but also by how much additional model size and training cost they require.

Training was carried out on RunPod infrastructure, and our design choices were shaped by the practical constraints of the competition setup. All reported runs were trained on RunPod instances with 8×H100 SXM GPUs. In particular, the submission artifact limit strongly influenced the size of the Rescue model, since even small increases in auxiliary parameters could determine whether a run remained eligible for submission.

5.2 Model sizes

We evaluate two Rescue-model sizes and multiple Navigator scales. For the Rescue model (Model B), we considered 0.5M parameters (524,288) and 1.0M parameters (1,048,576). The Navigator (Model A) base configuration contains approximately 17.06M parameters. We also study larger versions obtained by scaling the depth of the Navigator while keeping the latent dimension fixed:

- Base: 17,059,912 Navigator parameters;
- 2x: 33,596,048 Navigator parameters;
- 4x: 66,667,808 Navigator parameters.

In all main competitive experiments, the Rescue model remained fixed at 0.5M parameters, allowing us to isolate how the rescue effect behaves as the Navigator grows.

Variant	Params	Artifact size (MB)	Rescued BPB	Peak top-1 acc. (%)	Cross step	Notes
0.5M VRS	0.524M	15.93–15.94	1.8667 avg (1.8658 best)	50.51	3600	Fits limit
1.0M VRS	1.049M	16.18	1.8630	50.62	3400	Exceeds limit

Table 1: Short-run comparison of 0.5M and 1M Rescue variants, including peak nearest-neighbor top-1 accuracy (`nn_acc`) and the first stable BPB crossover step.

5.3 Training durations

To study both early-stage behavior and longer training dynamics, we evaluate the system across several time scales:

- 10-minute runs, including three 0.5M Rescue runs with different seeds and one 1M Rescue comparison run;
- a 41-minute run at base Navigator scale;
- a 1-hour 2x Navigator run;
- a 1-hour 4x Navigator run.

This setup allows us to examine whether the rescue effect appears early and consistently, whether it survives longer training, and whether it remains relevant when the main model is scaled substantially.

5.4 Optimization notes

During development, optimization details materially affected how the rescue effect appeared in practice. In particular, learning-rate behavior introduced instability in some early runs, making it important to distinguish between architectural conclusions and optimization artifacts.

The initial 10-minute runs used a time-based warmdown schedule with `WARMDOWN_ITERS` = 20000 and a 600-second wall-clock budget. When training was later extended to one hour, coupling the scheduler horizon to the longer wall-clock budget changed the effective Rescue-model learning-rate trajectory and led to late instability. The key correction was therefore to decouple total training time from the wall-clock used by the scheduler.

For the corrected extended runs, training was allowed to continue for up to `MAX_WALLCLOCK_SECONDS` = 3600, but the learning-rate schedule used `LR_WALLCLOCK_SECONDS` = 600, matching the original 10-minute schedule. After the warmdown reached the end of that 600-second schedule, the learning rate was held at a small floor using `LR_MIN_SCALE` = 0.02 rather than decaying to zero. In the main extended runs we therefore used `MAX_WALLCLOCK_SECONDS` = 3600, `LR_WALLCLOCK_SECONDS` = 600, `WARMDOWN_ITERS` = 20000, `LR_MIN_SCALE` = 0.02, and `RESCUER_LR` = 0.04. This preserves the stable early optimization dynamics of the 10-minute runs while allowing continued low-learning-rate refinement.

No Periodic Rescuer Reset, Rescuer Catch-Up, or detached-Rescuer training was used in the corrected runs reported here. Navigator and Rescuer were trained jointly throughout with the same MSE objective.

6 Results

6.1 Early rescue effect under tight parameter constraints

We first compare the 0.5M and 1M Rescue variants in short 10-minute runs. The 1M Rescue achieved a minimum validation BPB of 1.8630, while the 0.5M Rescue averaged 1.8667 across three seeds (1.8679, 1.8664, and 1.8658). The difference is only 0.0037 BPB, indicating that most of the rescue benefit is already captured by the smaller model.

Across the three 10-minute 0.5M runs, the average rescued BPB was 1.8667, compared with an average raw-path BPB (`val_bpb_A`) of 1.9401 within the jointly trained system, for an average gain of 0.0734 BPB. This shows that the Rescue path produces measurable improvements almost immediately, despite adding only a very small number of parameters relative to the Navigator.

This result was especially important in the Parameter Golf setting, since the 0.5M Rescue remained within the artifact limit while preserving almost all of the observed rescue benefit.

To better understand when the Rescue model begins to help in a meaningful way, it is useful to examine the *effective crossover* point. We define stable crossover as the first validation step after the initialization transient at which rescued BPB remains below the internal raw-path BPB for the subsequent validation window. In the corrected 0.5M short-run mean, this crossover occurs at step 3600 (2.68 minutes), while the 1M short run crosses slightly earlier at step 3400 (2.55 minutes).

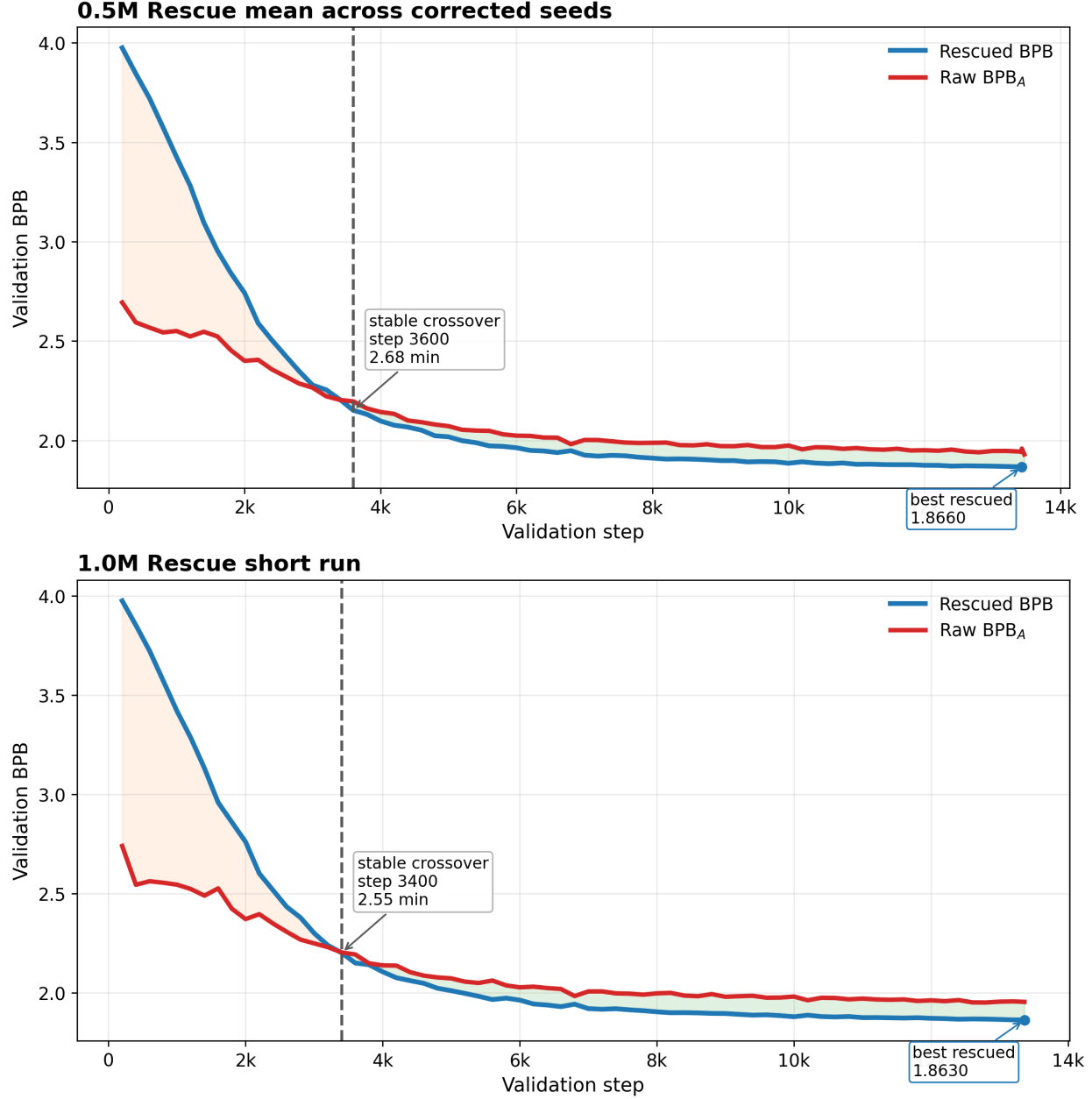


Figure 2: Short-run dynamics with explicit crossover markers. The top panel shows the corrected 0.5M mean across seeds, and the bottom panel shows the 1M short run. In both cases, blue denotes rescued decoding, red denotes raw-path decoding within the jointly trained system, orange shading indicates the interval where raw decoding is still better, and the dotted line marks the first stable crossover after the initialization transient.

6.2 Longer training at base scale

At longer training durations, the rescue effect remains present, though its magnitude depends on the training stage and model scale. In the 41-minute run, the full rescued system reached 1.8480

Setting	Nav. params	Rescue params	Duration	BPB	BPB _A	Gain	Peak nn_acc (%)	Cross step
Base avg	17.06M	0.524M	10 min	1.8667	1.9401	0.0734	50.51	3600
Base	17.06M	0.524M	41 min	1.8480	1.9091	0.0611	51.04	3600
2x	33.60M	0.524M	1 h	1.7570	1.8913	0.1343	53.35	2600
4x	66.67M	0.524M	1 h	1.7733	2.4144	0.6411	55.34	1600

Table 2: Summary of corrected runs. Here BPB denotes rescued decoding, BPB_A denotes internal raw-path decoding within the jointly trained VRS system, and nn_acc denotes nearest-neighbor top-1 accuracy.

BPB, while the internal raw path reached 1.9091 BPB, giving a rescue gain of 0.0611 BPB.

This result suggests that the Rescue model continues to improve decoding quality beyond the earliest phase of training, even after the Navigator itself has become substantially more informative. In the 41-minute run, the effective crossover occurs at step 3600 (2.68 minutes), indicating that the Rescue model becomes reliably useful very early relative to the full training duration.

6.3 Scaling the Navigator

The rescue effect becomes especially interesting under scaling. In the 2x Navigator one-hour run, the rescued system achieved the best absolute BPB among all VRS runs: 1.7570, compared with 1.8913 for the internal raw path. This corresponds to a gain of 0.1343 BPB.

In the 4x Navigator one-hour run, the rescued system achieved 1.7733 BPB, while the internal raw path remained much worse at 2.4144 BPB, for a very large gain of 0.6411 BPB. Notably, this improvement was produced by the same 0.5M Rescue model, even though the full system had grown to roughly 67.2M total parameters.

These results indicate that the Rescue model does not become irrelevant as the Navigator scales. In our experiments, the largest rescue gap appears in the largest model we tested, suggesting that larger Navigators may contain substantial token-predictive information that still remains poorly aligned for direct decoding.

The crossover analysis reinforces this interpretation. In the corrected 2x run, rescued decoding overtakes raw BPB_A at step 2600 (3.80 minutes), and in the 4x run it does so at step 1600 (4.77 minutes). Although the largest model crosses later in wall-clock time because each step is more expensive, it crosses at a smaller training step count and then opens the largest performance gap of all corrected runs.

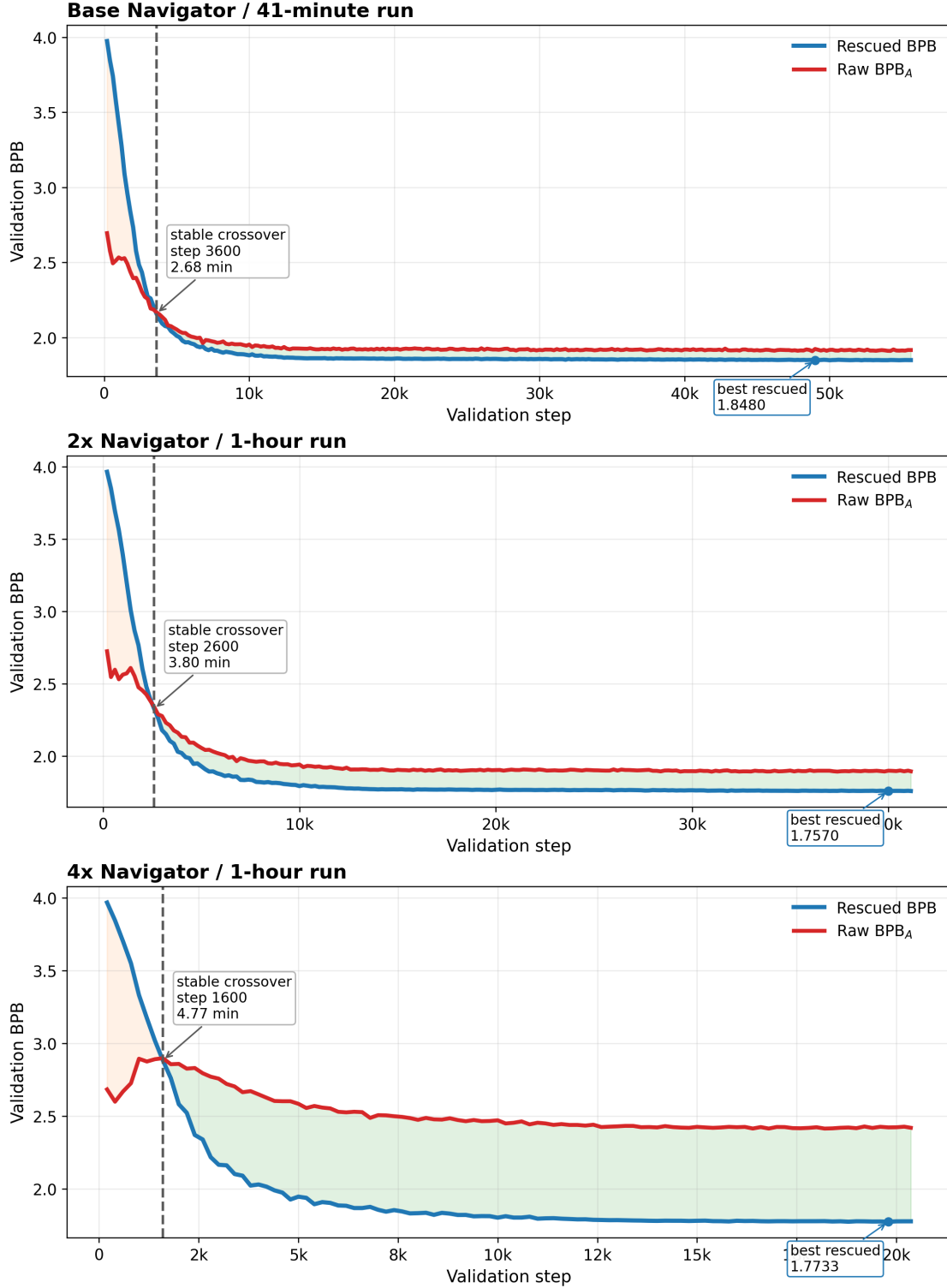


Figure 3: Large-scale dynamics with explicit crossover markers. Each panel compares rescued BPB against the internal raw-path BPB_A and marks the first effective crossover after the initialization transient. These plots make clear that the Rescue model becomes useful early in training and remains better than raw decoding thereafter.

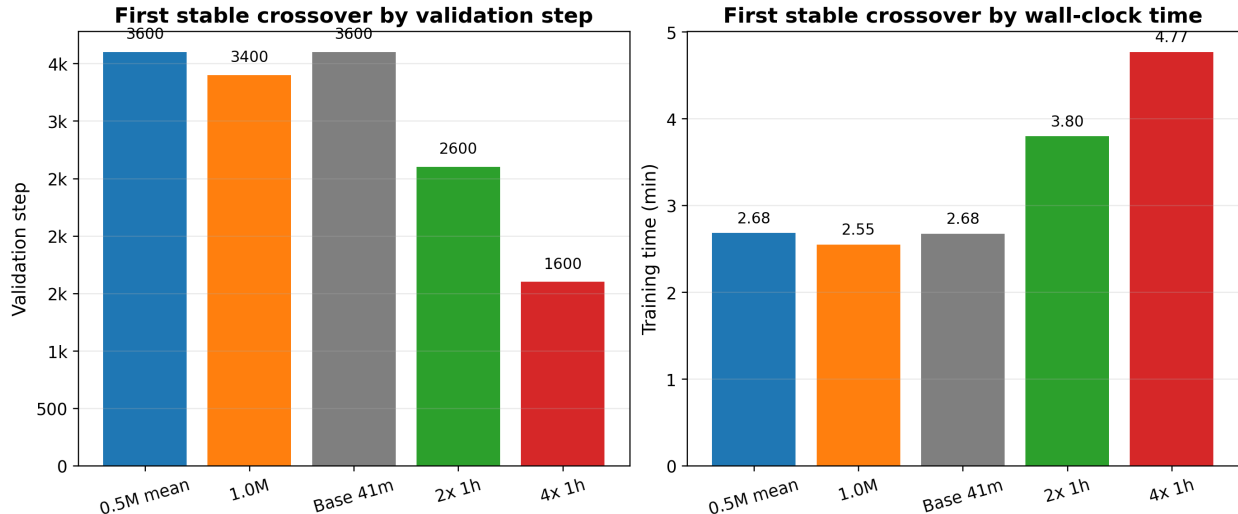


Figure 4: Summary of first effective crossover points across runs. Left: crossover step. Right: crossover time in minutes. The 0.5M short-run mean and the base 41-minute run both cross at step 3600, the 1M run at step 3400, the 2x run at step 2600, and the 4x run at step 1600.

6.4 Top-1 recovery versus full probabilistic quality

The contrast between `nn_acc` and BPB clarifies that top-1 recovery and full probabilistic calibration are distinct. The 4x run achieved the highest `nn_acc` observed in our experiments (0.5534), yet its best BPB (1.7733) was still slightly worse than the 2x run (1.7570). VRS appears to improve both token ranking and distributional quality, but not always by the same amount.

6.5 Main empirical takeaway

Across short runs, longer runs, and scaled Navigators, VRS consistently improves over raw decoding from v_{void} . The 0.5M Rescue model is nearly as effective as the 1M variant while fitting the artifact limit, and the rescue gap can grow with Navigator scale, reaching 0.6411 BPB in the 4x one-hour setting.

6.6 Comparison against external regression-only baselines

To complement the internal raw-path probe (`val_bpb_A`), we also evaluated three separately trained regression-only baselines without VRS under comparable 10-minute short-run conditions. These runs provide a more meaningful external reference than `val_bpb_A`, since they measure a true Navigator-only regression system rather than the raw path inside a jointly trained VRS model.

Model	Duration	Seeds	Min BPB	Peak <code>nn_acc</code>	Artifact MB
Regression-only baseline	10 min	3 seeds	2.0941–2.1301	0.5005–0.5018	15.19–15.23
VRS (0.5M, 10 min)	10 min	3 seeds	1.8658–1.8679	0.5032–0.5067	15.93–15.94

Table 3: Comparison between three separately trained 10-minute regression-only baselines and three 10-minute 0.5M VRS runs. Unlike `val_bpb_A`, which is an internal raw-decoding probe of a jointly trained VRS system, these baselines provide an external reference for regression without a rescue module.

Figure 5 compares the stepwise BPB trajectories of the three 10-minute VRS runs against the three external regression-only baselines in the same short-run regime.

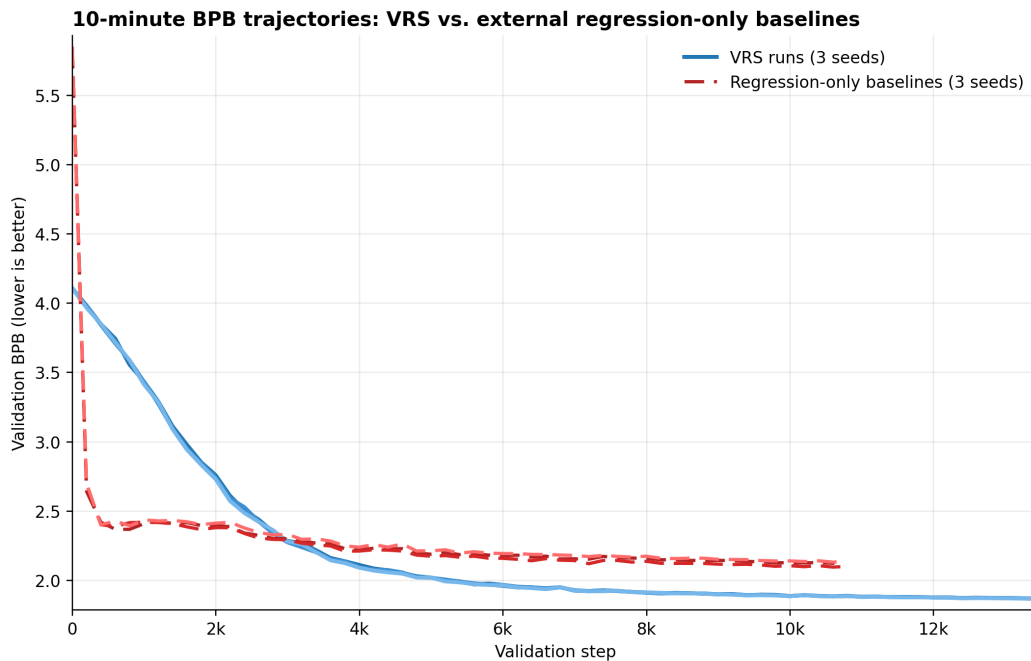


Figure 5: Stepwise validation BPB for the three 10-minute 0.5M VRS runs and the three 10-minute external regression-only baselines. The blue solid curves show rescued BPB over validation steps, while the red dashed curves show the separately trained regression-only baselines. The separate classification baseline present elsewhere in the workbook is intentionally excluded. Lower is better.

Across the three baseline seeds, minimum validation BPB ranged from 2.0941 to 2.1301, averaging 2.1152, while peak `nn_acc` ranged from 0.5005 to 0.5018, averaging 0.5012. By contrast, the 10-minute 0.5M VRS runs reached 1.8658–1.8679 BPB, averaging 1.8667, with peak `nn_acc` of 0.5032–0.5067 (average 0.5051). Thus, VRS remains clearly better even when compared against a real external regression baseline rather than only against the internal raw-path probe.

The improvement pattern is also informative. Relative to the regression-only baselines, VRS improves BPB by a large margin, while improving `nn_acc` only slightly. This is consistent with our interpretation that the Rescue path contributes more strongly to full distributional quality and target scoring than to top-1 recovery alone: under raw regression, the correct token is often already near the top, but VRS appears to improve its relative score and probability under the full decoded

distribution.

6.7 Qualitative geometry and small ablations

Figure 6 provides a qualitative visualization of the Rescue effect. For selected probe positions, the Navigator output v_{void} lies far from the corresponding target embedding, while the rescued output v_{rescued} is substantially closer. Although PCA compresses the original 512-dimensional geometry and makes rescued vectors appear more clustered than they truly are, the underlying distance reductions are computed in the full embedding space. These plots therefore illustrate latent realignment rather than BPB directly.

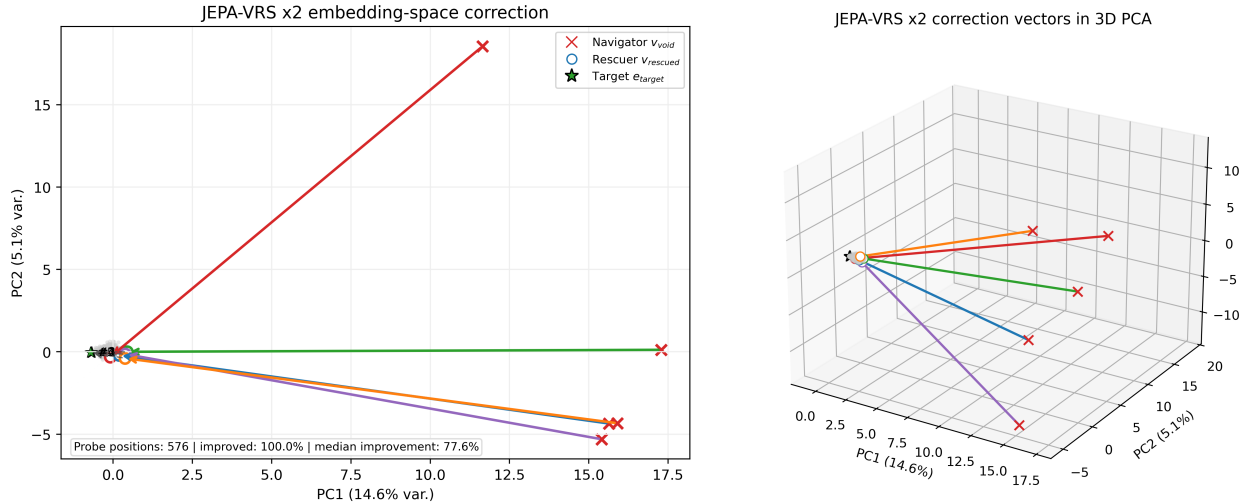


Figure 6: Qualitative rescue-geometry visualization. Left: 2D PCA view of target embeddings, raw Navigator outputs, and rescued outputs for selected probe positions. Right: complementary 3D PCA view of the same effect. Distances discussed in the text are computed in the full embedding space, not in PCA space.

When Model A is frozen, Model B still improves BPB and accuracy slightly, but much less than in the jointly trained setting, and performance soon plateaus. This suggests that Model B can rapidly exploit much of the decodable structure present in a fixed latent space, but that the larger rescue gains depend on continued co-adaptation between Model A and Model B. In this view, the Rescue path is better understood as a jointly learned alignment mechanism than as a strong standalone post-hoc rescuer.

In a single-seed ablation, replacing the 0.5M Rescue MLP with a linear layer of only approximately 0.26M parameters preserved the rescue effect and slightly improved BPB in the tested configuration. Although this result is preliminary, it suggests that part of the raw decoding gap may arise from an approximately linear misalignment between v_{void} and the shared token-embedding geometry.

7 Discussion

Our results suggest that regression-based token prediction may involve a partial separation between contextual computation and lexical decoding geometry. In the present setup, VRS is best understood as a small jointly learned alignment head: the Navigator appears to do most of the contextual work, while VRS improves how that information is expressed in token space. We interpret this as an exploratory result about how transformers can behave under regression training, not as evidence that regression should replace standard classification objectives.

Recent work has also begun to explore continuous next-vector prediction directly in language models, suggesting that output-space design is an active research direction rather than an isolated curiosity [11].

This does not mean that the Rescue model is an independent language model; rather, it behaves like a small geometric decoder attached to an already informative latent. Taken together, the frozen-A and linear-rescuer ablations are consistent with this interpretation: some of the correction appears approximately linear, but the larger gains depend on continued co-adaptation with the Navigator. If this picture holds more broadly, similar auxiliary decoders might be reusable across latent predictors that share sufficiently similar embedding geometry.

8 Limitations

While the void interpretation is consistent with our results, it remains a modeling hypothesis rather than a fully established geometric characterization of embedding space. We use the term *void* as a formal concept within this paper, but we do not claim to have mathematically proven the existence or exact structure of such regions.

A key limitation is that `val_bpb_A` is an internal raw-decoding probe of a jointly trained VRS system rather than a separately trained Navigator-only baseline. We now partially address this with three short-run external regression-only baselines, but we do not yet report matched external baselines across longer durations, larger Navigator scales, or alternative optimization settings. Our experiments also do not yet establish whether the Rescue model is transferable across seeds, model sizes, or independently trained Navigators. Although the Rescue effect is consistently visible in the settings we studied, we have not shown that a Rescue model trained in one run can be reused successfully in another.

Similarly, we do not claim that the Rescue model is universal, nor that it functions as an independent language model. The Rescue model receives only v_{void} , has no direct access to context or positional structure, and therefore should not be interpreted as performing full language modeling on its own.

Another limitation is that `nn_acc` captures only top-1 recovery, not full distributional calibration. A model may improve nearest-neighbor decoding while still assigning weak probability mass to the correct token, and vice versa. For this reason, the gap between BPB and top-1 accuracy remains an important but only partially characterized part of our interpretation.

Finally, the current study focuses on a specific regression-based language modeling setup under Parameter Golf constraints. We therefore do not make a global claim that regression is better or

worse than standard classification-based language modeling. Additional experiments will be needed to determine how broadly these findings extend beyond this particular architecture, scale range, and training regime.

9 Future Work

Several directions could strengthen and clarify the claims made in this paper.

First, a richer evaluation suite would help determine whether rescue primarily improves ranking, calibration, or both. In particular, metrics such as top-5 accuracy, target probability, and logit margin would provide a more detailed view of how rescued decoding differs from raw decoding.

Second, transfer experiments could test whether the Rescue model learns a run-specific correction or something more general. Promising directions include training a Rescue model on one seed and evaluating it on another, freezing the Rescue model and swapping in a different Navigator, and measuring rescue behavior across different model sizes.

Third, it would be useful to broaden the current adapter comparison beyond the preliminary ablations reported here. In particular, residual correction layers and slightly more expressive but still lightweight adapters could help determine how much of the observed effect truly depends on nonlinearity versus simple reprojection.

Together, these directions could clarify whether the Void Rescue System is best understood as a local correction module, a reusable lexical decoder, or the first instance of a broader geometric decoding framework situated alongside emerging JEPA-style and continuous-language approaches [5, 11].

10 Conclusion

We introduced the Void Rescue System, a lightweight auxiliary decoder for regression-based token prediction. Across short runs, longer runs, and scaled Navigators, it consistently improves BPB and nearest-neighbor accuracy over raw decoding, with a 0.5M model performing nearly as well as a 1M version while staying within artifact limits.

These results support the view that regression latents can be informative before they are directly decodable, and that part of the challenge in regression-based language modeling is not generating more information but aligning that information for decoding. More broadly, the paper is intended as an exploration of regression as a way to study transformer representations and decoding behavior, rather than as a head-to-head argument that regression is superior to classification.

Acknowledgments

I thank RunPod and OpenAI for the \$500 in RunPod credits that made these experiments possible. I also acknowledge substantial help from ChatGPT, Codex, Claude, Claude Code, Prism, and Gemini during experimentation and drafting.

Author’s Note

This paper was developed independently as a high school student using RunPod credits. The claims are intentionally narrow and the results preliminary, but they represent a concrete line of inquiry I intend to develop further.

References

- [1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.
- [2] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. Breaking the softmax bottleneck: A high-rank RNN language model. *arXiv preprint arXiv:1711.03953*, 2017.
- [3] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain, 2017. Association for Computational Linguistics.
- [4] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael G. Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. *arXiv preprint arXiv:2301.08243*, 2023.
- [5] Hai Huang, Yann LeCun, and Randall Balestriero. LLM-JEPA: Large language models meet joint embedding predictive architectures. *arXiv preprint arXiv:2509.14252*, 2025.
- [6] Sachin Kumar and Yulia Tsvetkov. Von mises-fisher loss for training sequence to sequence models with continuous outputs. *arXiv preprint arXiv:1812.04616*, 2018.
- [7] Evgeniia Tokarchuk and Vlad Niculae. On target representation in continuous-output neural machine translation. In *Proceedings of the 7th Workshop on Representation Learning for NLP*, pages 227–235, Dublin, Ireland, 2022. Association for Computational Linguistics.
- [8] Evgeniia Tokarchuk and Vlad Niculae. The unreasonable effectiveness of random target embeddings for continuous-output neural machine translation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 653–662, Mexico City, Mexico, 2024. Association for Computational Linguistics.
- [9] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*, 2019.
- [10] Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Nearest neighbor machine translation. *arXiv preprint arXiv:2010.00710*, 2020.
- [11] Chenze Shao, Darren Li, Fandong Meng, and Jie Zhou. Continuous autoregressive language models. *arXiv preprint arXiv:2510.27688*, 2025.