Optimizing Hidden Markov Language Models: An Empirical Study of Reparameterization and Initialization Techniques

Ivan Lee and **Taylor Berg-Kirkpatrick**
UC San Diego
{iylee,tberg}@ucsd.edu

Abstract

Hidden Markov models (HMMs) are valuable for their ability to provide exact and tractable inference. However, learning an HMM in an unsupervised manner involves a non-convex optimization problem that is plagued by poor local optima. Recent work on scaling HMMs has shown this challenge only intensifies as the number of hidden states grows. We provide a comprehensive empirical analysis of two approaches to enhance HMM optimization: reparameterization and initialization of HMM transition and emission parameters using neural networks. Through extensive experiments on language modeling, we find that (1) these techniques enable effective training of large-scale HMMs, (2) simple linear reparameterizations of HMM parameters perform as well as more complex neural ones, and (3) the two approaches are complementary, yielding the best results when combined.

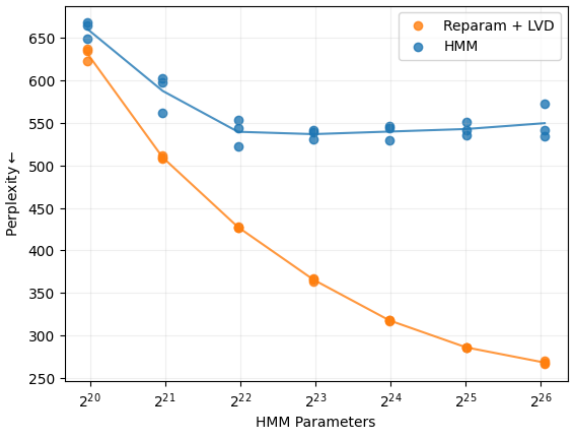

Figure 1: Perplexity as a function of HMM parameters. (Blue) Scaling the number of hidden states, when using random initialization and standard parameterization, often converges to suboptimal local optima. (Orange) Reparameterization and improved initialization alleviates this issue.

1 Introduction

Despite being largely supplanted by more performant neural networks (NNs), HMMs remain valuable in modern applications due to their ability to provide exact and tractable inference. These applications include data synthesis (Xie et al., 2021; Lavie et al., 2024), controllable text generation (Zhang et al., 2023), non-autoregressive translation (Li et al., 2024), aligning speech and text for low-resource languages (Ljubevsi'c et al., 2024), Chinese spelling correction (Wang et al., 2024), and data analysis (Qi and Inaba, 2024).

Recent work showcased the expressive capacity of HMMs for language modeling by massively scaling up the number of hidden states using modern compute hardware (Chiu and Rush, 2020). The results were promising, highlighting a possibly lighter-weight and more interpretable alternative to neural LMs for certain applications. However, the core optimization problem behind unsupervised learning in HMMs remains a challenge: the marginal likelihood learning objective is non-convex, and current optimization techniques tend towards poor local optima. Indeed, Liu et al. (2022) found that as the number of hidden states grows, this issue becomes more pronounced, and larger HMMs often fail to utilize their expanded capacity. Random restarts offer an approach to mitigate this problem (Berg-Kirkpatrick and Klein, 2013), but are computationally expensive.

In contrast, the training of neural models rarely suffers from poor local optima (Nguyen and Hein, 2017), despite also involving non-convex optimization. *Can insights from neural training be leveraged to improve learning in HMMs?* We study this question empirically by comprehensively investigating extensions and combinations of two techniques: *neural reparameterization* (NR) and *neural initialization* (NI). It has been hypothesized that the tendency of neural optimization to avoid poor local optima is due to the shape of neural parameterizations (Zhang et al., 2021). Might the reparam-

Findings of the Association for Computational Linguistics:
NAACL 2025, pages 7727–7738

eterization of HMMs using neural networks allow HMM learning to inherit some of these benefits?

Chiu and Rush (2020) leveraged NR of traditional multinomial HMM parameters, but primarily to reduce memory consumption and did not extensively evaluate whether this approach also improved optimization and why. Here, we seek to directly measure the effects of reparameterization on HMM optimization and to compare different reparameterization structures. Similarly, it has long been known that smart heuristic initializations can aid in non-convex optimization (Klein and Manning, 2004). Recently, however, neural pretraining techniques have been proposed as a new method for the initialization of HMMs and other probabilistic models (Liu et al., 2022). We aim to empirically study whether neural initialization techniques can be effectively combined with NR.

Our experiments demonstrate that the combination of both techniques can indeed yield substantially improved optimization performance in HMMs. For example, as depicted in Figure 1, we find that our best approach demonstrates gains in perplexity with larger HMMs, whereas traditional training techniques exhibit reduced performance as the HMM grows. Furthermore, our experimental analysis reveals that the benefits of NR are almost fully explained by simple linear reparameterization strategies, suggesting takeaways for related probabilistic non-convex problems from other model classes.

2 Experimental Setup

Following Chiu and Rush (2020) and Liu et al. (2022), we focus on the task of language modeling. We train HMMs with various initialization and reparameterization strategies on 500M tokens and test on 1M held out tokens from the Slimpajama dataset (Soboleva et al., 2023). Our baseline model is a standard HMM with multinomial transition and emission distributions, randomly initialized (denoted as ★ in our figures). Unless otherwise stated, we fix the number of hidden states to 512 and sequence length to 64 tokens.

For optimization, we use minibatch SGD with the AdamW optimizer, employing a linear warmup followed by cosine decay of the learning rate (additional details are provided in Appendix 5). While HMMs are traditionally trained with the Baum-Welch (BW) algorithm, our preliminary experiments found that SGD consistently outperformed

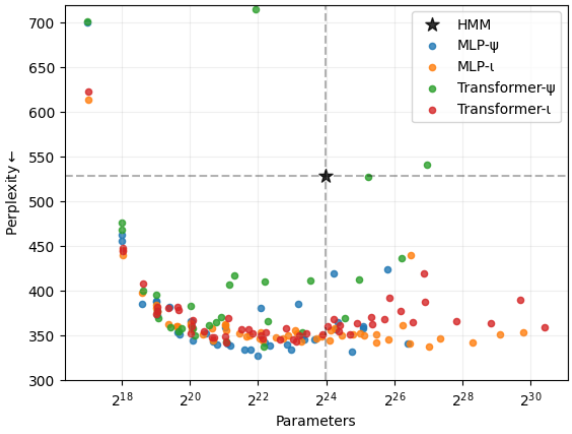

Figure 2: Perplexity as a function of reparameterization capacity and architecture design (Section 3). ★ indicates our baseline: a randomly initialized HMM with standard parameterization, optimized with SGD.

BW[1] in both training time and final perplexity. We train for a single epoch and report test perplexity, though we observe nearly identical results on the training set.

Our experiments systematically explore two approaches to improve HMM optimization: First, we study neural reparameterization (Section 3), where we replace the HMM's multinomial parameters with the output of a neural network. We investigate how the capacity and architecture of the neural network affect optimization and examine whether the benefits can be achieved with simpler linear parameterizations. Next, we explore initialization strategies (Section 4), where we use pretrained language models to guide parameter initialization of an HMM. In particular, we study one form of initialization called latent variable distillation (Liu et al., 2022) and ask two key questions: (1) how does the strength of the teacher model (measured by its training data size) affect HMM performance, and (2) what is the optimal amount of compute to dedicate to initialization before diminishing returns set in? Finally, we investigate the combination of both approaches (Section 5), examining whether their benefits are complementary.

3 Reparameterization

Neural reparameterization (NR) (Tran et al., 2016; Chiu and Rush, 2020) replaces an HMM's emission and transition matrices with the output of a neu-

[1]Given the size of our dataset, we employed mini-batch BW: $\theta_{t+1} \leftarrow \alpha \cdot \theta^{\text{new}} + (1-\alpha) \cdot \theta_t$ where α is the learning rate and θ^{new} are the updated parameters returned by one step of BW.

ral network. For example, Chiu and Rush (2020) reparameterize an HMM with the multilayer perceptron (MLP) architecture depicted in Appendix 10. While they primarily employed NR to reduce parameter count for training large HMMs, they also observed improvements in perplexity—motivating our deeper investigation into why and how NR enhances HMM optimization. We explore the impact of NR with respect to three key aspects: the capacity of the neural network, choice of architecture (e.g., MLP versus Transformer), and functional form (e.g. linear versus non-linear).

Does NR scale matter? Given that Chiu and Rush (2020) were primarily interested in NR for training efficiency, we further explore the relationship between perplexity and the NN's capacity. When clear from context, "the NN" refers to the neural network used to reparameterize an HMM. In particular, we ask: does overparameterizing the NN lead to improved local optima?

Figure 2 illustrates the impact of reparameterizing an HMM using NNs with varying parameter counts and architectures. We defer discussion of architecture choice as is not crucial for understanding the following takeaways: NR consistently enhances perplexity compared to the baseline ★. This improvement is evident even when the NN is considerably smaller than the HMM it reparameterizes. However, increasing the NN size beyond 2^{22} parameters does not yield additional benefits, indicating that scaling the NN does not parallel the advantages of scaling the HMM's hidden states. Conversely, reducing the NN capacity below 2^{22} parameters predictably worsens perplexity. These findings also hold true for hidden state sizes other than 512 as shown in Appendix 11 and 12.

Does NR architecture matter? We now consider the impact of the NN's architecture, focusing on both MLPs and transformers (TRFs), as well as their *monolithicity*—whether the architecture uses a single neural network instance or multiple instances. For example, we consider the design of Chiu and Rush (2020) as being non-monolithic (ψ) since it uses three separate neural networks (f_{in}, f_{out}, and f_{emit}), each taking the hidden state embedding H as input. In contrast, a monolithic (ι) design employs just a single neural network instance f that processes H. An illustrative example comparing these designs can be found in Appendix 13.

Our findings, presented in Figure 2, indicate that any architectural choice significantly improves performance over the baseline. The non-monolithic MLP design by Chiu and Rush (2020) often achieves the best perplexity scores, only being surpassed by its monolithic counterpart at scales above 2^{22} parameters. However, this limitation is not practically relevant since we observe no further perplexity improvements beyond this scale. When comparing different architectures, monolithic MLPs and TRFs exhibit similar performance for most parameter sizes, though MLPs gain an advantage for sizes greater than 2^{25}. The worst performing configuration was the non-monolithic transformer design.

Perplexity ↓	Mean	Min	Max
Ablation A (single H)	711	710	713
Ablation B (split H)	**348**	**346**	**350**
HMM ★	540	529	546
Chiu and Rush ×	357	355	359

Table 1: Linear reparameterizations are competitive with, and often superior to, their neural counterparts.

Does NR functional form matter? We now ablate Chiu and Rush (2020)'s NR to identify which components contribute to performance gains. We begin with the simplest configuration, referred to as ablation A, where all instances of MLPs are removed, resulting in a design that prohibits asymmetric transition probabilities (Appendix 14, left). Given this limitation, we also consider ablation B, where the hidden state embeddings are split three ways (Appendix 14, right). We find that ablation B surpasses Chiu and Rush (2020)'s NR (Table 1). This is surprising because ablation B is simply a low-rank factorization of the transition and emission matrices: $a = H_{in}H_{out}^{\intercal}$, $b = H_{emit}V^{\intercal}$ where $H_i \in \mathbb{R}^{n \times d}$ and $d < n$, implying that the benefits of Chiu and Rush (2020)'s NR seem to arise mainly from factorizing the HMM's parameters, rather than any properties unique to NNs.

We also examine the impact of independently factorizing each HMM matrix and find that while factorizing all matrices yield the best results, most improvements come from the emission matrix. We then further ablate Chiu and Rush (2020)'s NR to identify which neural components offered value, discovering a few more performant, though minor, configurations. See Appendix A for details.

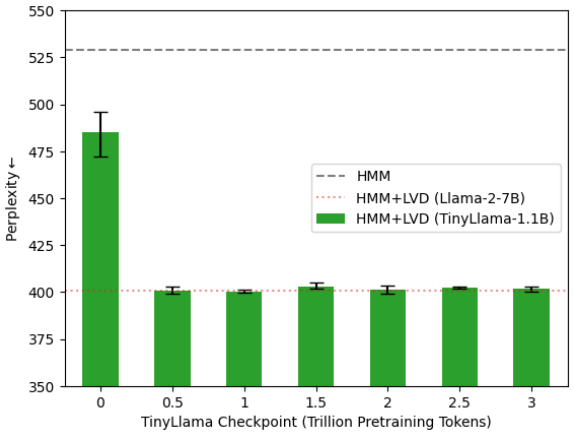

Figure 3: Impact of various teachers on LVD. All teachers lead to improvements over the baseline, including a randomly initialized one, albeit to a lesser extent. There is no correlation between the teacher's strength and the extent of improvement.

4 Initialization

Latent Variable Distillation (LVD) (Liu et al., 2022) uses a pretrained NN (teacher) to initialize the parameters of an HMM (student), hoping to form a better starting point which will lead to better local optima. More specifically, each observation $o \in O$ is mapped to a hidden state $s \in S$ using the NN's contextualized embedding of o (a token in the language modeling setting). The underlying idea is that observations generated from the same hidden state should be positioned closely together in the embedding space. This is implemented by applying the k-means algorithm to cluster the embeddings, where $k = |S|$. The weights of the HMM are initialized by maximizing the joint distribution $P(O, S)$ for a predetermined number of steps. This is followed by standard unsupervised training, i.e., maximizing the marginal likelihood $P(O)$. See Appendix 19 for pseudocode of this process.

We now examine the effects of LVD initialization on HMM training. Liu et al. (2022) show that LVD results in easier-to-optimize HMMs, but several questions persist. We aim to clarify the link between the strength of the teacher NN and the student HMM. Additionally, we seek to determine the optimal number of LVD steps before improvements plateau. Note that we do not use NR in the following experiments, meaning we directly update the emission and transition matrices of the HMM using SGD.

Does Teacher Strength Matter? We train several HMMs using LVD while varying the strength of the teacher. Specifically, we utilize `TinyLlama-1.1B` (Zhang et al., 2024) checkpoints, which range from 500B to 3T pretraining tokens. For the strongest teacher, we employed `Llama-2-7B` (Touvron et al., 2023). As illustrated in Figure 3, our findings confirm that LVD facilitates easier optimization of HMMs. However, we observed two surprising results. First, the strength of the teacher has minimal impact: `TinyLlama` trained on 500B tokens performed comparably to `Llama-2`, which is seven times larger and trained on 2T tokens. Second, using LVD with a *randomly initialized teacher* still offers advantages over not using LVD at all.

Effect of Number of Initialization Steps Again, we train several HMMs with LVD but now vary the number of training steps dedicated to maximizing the joint probability $P(O, S)$ before resuming unsupervised training by maximizing the marginal likelihood $P(O)$. We find that initializing with LVD saturates after 5M tokens. However, leveraging LVD throughout the entire training process worsens performance compared to not using LVD. We visualize our findings in Appendix 8.

5 Reparameterization + Initialization

We now consider approaches that incorperate both reparameterization and initialization.

LVD + Reparameterization The benefits of LVD and NR are impressive on their own, but do they complement each other or simply lead the HMM to the same local optima? Figure 4 demonstrates that these methods are indeed complementary, regardless of the architecture used. Additionally, LVD reduces the variance in training runs, a benefit not observed with NR alone.

Transfer Learning A natural extension of NR is to initialize the NN with the weights of a pretrained NN. To test this, we train a decoder-only TRF language model T on 4B tokens. We then reparameterize an HMM with a monolithic TRF and initialize it with the weights of T. We find that this form of transfer learning is beneficial as long as the learning rate (for training the HMM) is properly tuned. We also experiment with initializing the TRF using checkpoints of `TinyLlama-1.1B`, an LM many orders of magnitude larger than the HMM it reparameterizes. We reconfirm the efficacy of transfer learning as all checkpoints outperformed a randomly initialized TRF. However, contrary to our intuition, initializing with a stronger checkpoint did not yield better results. We visualize our findings

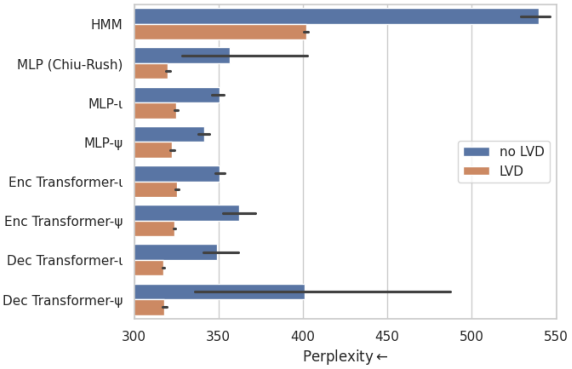

Figure 4: LVD improves perplexity and reduces variance across random restarts, regardless of architecture. The benefits of LVD and NR are complementary.

in Appendix 6 and 7.

Transfer Learning + LVD Given that transfer learning and LVD are both initialization methods, are their benefits redundant? To explore this, we compared two HMMs reparameterized with a monolithic TRF: one initialized randomly and the other with `TinyLlama`. Both were trained using LVD for 10M tokens before continuing with standard training. We find that combining transfer learning with LVD does not yield significant improvements over using LVD alone. See Appendix 4 for full results.

6 Conclusion

We conducted an empirical investigation into the use of neural initialization and reparameterization to address the challenges of learning HMMs. Our study confirmed the effectiveness of each approach and revealed that they complement each other. Additionally, we discovered that the advantages of neural reparameterization can be largely attributed to a low-rank linear factorization of the HMM.

7 Limitations

While this empirical analysis aimed to be as comprehensive as possible, we could not test every possible experiment setting. It is always possible that increasing the number of random restarts and expanding our hyperparameter search could reveal different results.

References

Taylor Berg-Kirkpatrick and Dan Klein. 2013. Decipherment with a million random restarts. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 874–878, Seattle, Washington, USA. Association for Computational Linguistics.

Justin T Chiu and Alexander M. Rush. 2020. Scaling hidden markov language models. In *Conference on Empirical Methods in Natural Language Processing*.

Dan Klein and Christopher Manning. 2004. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 478–485, Barcelona, Spain.

Itay Lavie, Guy Gur-Ari, and Zohar Ringel. 2024. Towards understanding inductive bias in transformers: A view from infinity. *ArXiv*, abs/2402.05173.

Haoran Li, Zhanming Jie, and Wei Lu. 2024. Non-autoregressive machine translation as constrained HMM. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 12361–12372, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Anji Liu, Honghua Zhang, and Guy Van den Broeck. 2022. Scaling up probabilistic circuits by latent variable distillation. *ArXiv*, abs/2210.04398.

Nikola Ljubevsi'c, Peter Rupnik, and Danijel Korvzinek. 2024. The parlaspeech collection of automatically generated speech and text datasets from parliamentary proceedings.

Quynh Nguyen and Matthias Hein. 2017. The loss surface of deep and wide neural networks. *Preprint*, arXiv:1704.08045.

Zhiyang Qi and Michimasa Inaba. 2024. Data augmentation integrating dialogue flow and style to adapt spoken dialogue systems to low-resource user groups. In *Proceedings of the 25th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 159–171, Kyoto, Japan. Association for Computational Linguistics.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama.

Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cristian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony S. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, A. V. Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew

Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288.

Ke M. Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. 2016. Unsupervised neural hidden Markov models. In *Proceedings of the Workshop on Structured Prediction for NLP*, pages 63–71, Austin, TX. Association for Computational Linguistics.

Xi Wang, Ruoqing Zhao, Jing Li, and Piji Li. 2024. An unsupervised domain-adaptive framework for chinese spelling checking. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.* Just Accepted.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2021. An explanation of in-context learning as implicit bayesian inference. *ArXiv*, abs/2111.02080.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2021. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM*, 64(3):107–115.

Honghua Zhang, Meihua Dang, Nanyun Peng, and Guy Van den Broeck. 2023. Tractable control for autoregressive language generation. *ArXiv*, abs/2304.07438.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *ArXiv*, abs/2401.02385.

a	b	π	Mean	Min	Max
False	False	False	607	561	681
True	False	False	814	750	871
True	False	True	865	752	950
False	True	False	345	344	347
True	True	False	**338**	**337**	**339**
True	True	True	**337**	**336**	**339**

Table 2: Effect of factorizing HMM parameters on perplexity (lower is better).

A Reparameterization Continued

We examined the impact of independently factorizing each HMM matrix (Table 2) and found that while factorizing all matrices yielded the best results, most improvements came from the emission matrix b. We then investigated whether any neural properties offered benefits. Specifically, we ablated Chiu and Rush (2020)'s NR by varying the number of linear layers, and the use of activation functions and layer normalization. We also assessed monolithicity (Figure 15) and splitting H into three separate hidden state embeddings (Figure 16). Although a few configurations surpassed ablation B (Table 3), the improvements were minimal.

B Initializing with Cluster Centroids

We examine the impact of initializing hidden state embeddings with the cluster centroids obtained during LVD and vocabulary embeddings with the teacher's language modeling head. Specifically, we use `TinyLlama` as the LVD teacher for initializing the embeddings of Chiu and Rush (2020)'s NR. Our findings are negative: both initialization methods degrade performance, even when embeddings are frozen for the initial 20 million training tokens. Results are shown in Figure 5.

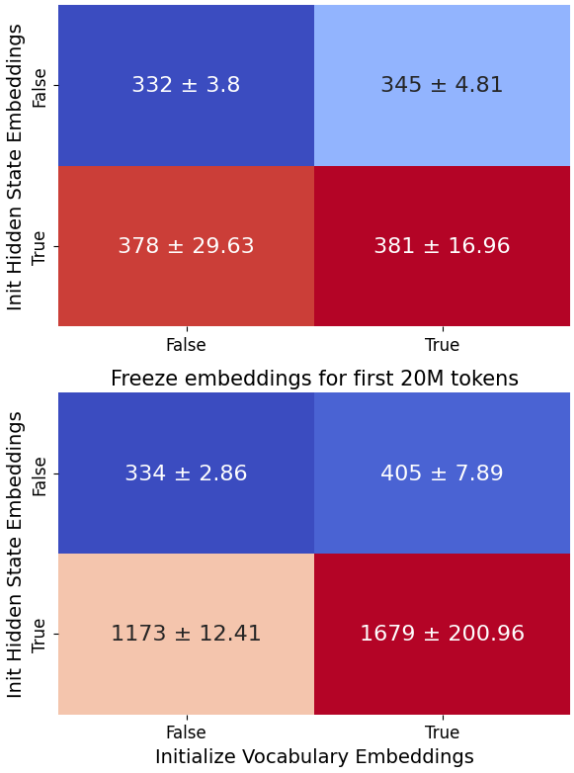

Figure 5: Clusters obtained from LVD are used to initialize hidden state embeddings. The language modeling head of the LVD teacher is used to initialize vocabulary embeddings.

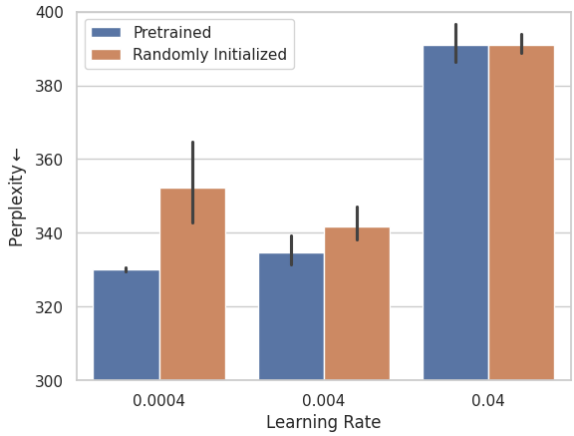

Figure 6: Initializing parameterization with a checkpoint pretrained on language modeling improves perplexity. However, high learning rates cancel out any benefit. 3 random seeds, error bars show min and max performance. Pretrained transformer was trained on language modeling on 4B tokens.

Split H	Monolithic	Linear Layers	Skip Connection	ReLU	LayerNorm	Perplexity_mean	Perplexity_min	Perplexity_max
False	False	1	False	True	False	329.5	322.7	340.9
True	False	1	False	True	False	336.6	331.7	341.5
False	False	2	True	True	False	336.7	332.0	342.7
True	False	2	False	False	True	341.2	339.0	343.6
True	False	2	True	True	False	342.6	334.3	353.5
True	True	2	True	True	False	343.5	337.1	350.2
False	False	1	False	False	False	344.4	342.1	348.1

Table 3: Variants of Chiu and Rush (2020)'s NR that outperformed ablation B in Table 1.

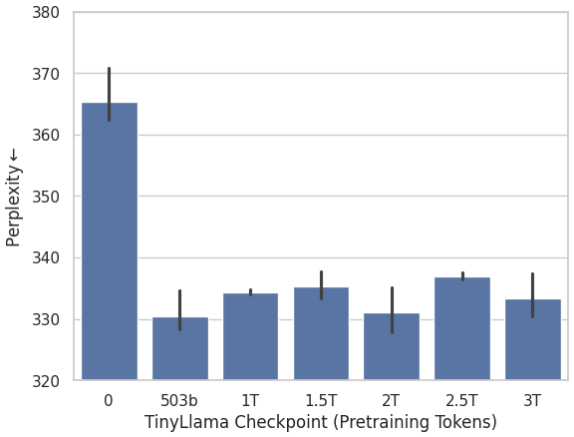

Figure 7: Parameterizing an HMM with TinyLlama, a 1.1B parameter transformer. Horizontal lines show the best performing runs from Figure 2. Initializing TinyLlama with pretrained checkpoints offers benefits over random initialization, however, more training tokens did not result in meaningful improvements. Substantially smaller, randomly initialized parameterizations perform just as well or better than TinyLlama.

Algorithm 1 Latent Variable Distillation

```
T, M ← teacher model, student model
Step 1: Learn clusters of embeddings
embeds ← []
for sequence in text-corpus-1 do
    for token in sequence do
        embeds.append(T(token))
    end for
end for
clusters ← k-means(embeds, k=hidden-states)
Step 2: T guides M to optimize P(O, S)
for sequence in text-corpus-2 do
    embeddings ← T(sequence)
    S ← get-labels(clusters, embeddings)
    M optimizes P(O, S)
end for
Step 3: M optimizes P(O)
for sequence in text-corpus-3 do
    M optimizes P(O)
end for
```

	Perplexity (Min-Max) ↓
Random Init.	365 (362-371)
+LVD	317 (316-317)
Pretrained Init.	333 (330-337)
+LVD	**313** (312-315)

Table 4: Mean perplexity. HMM is reparameterized with a 1.1B parameter transformer using the ι template. We use TinyLlama-1.1B trained on 3T tokens as the checkpoint for LVD and reparameterization.

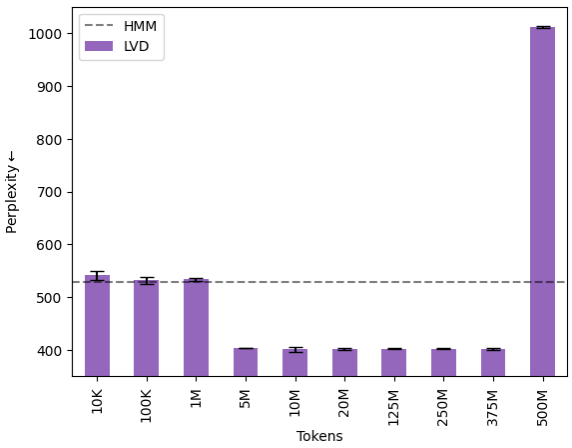

Figure 8: Perplexity versus the number of tokens used to optimize $P(O, S)$, guided by latent variable distillation (LVD). Following LVD, optimization of $P(O)$ continues until a cumulative total of 500M tokens is reached. Benefits of LVD emerge only with a minimum of 1M tokens, achieving full potential at 5M tokens. Extending LVD throughout the entire training period, however, proves counterproductive (rightmost bar).

Parameter	Argument
Training Task	Language modeling
Training Data	500 million tokens from Slimpajama
Test Data	1 million tokens from Slimpajama
Tokenizer	Llama 2 tokenizer with vocabulary size of 32,000
Precision	bf16-mixed
Batch Size	200
Sequence Length	64 tokens
Optimization Strategy	Minibatch Stochastic Gradient Descent
Optimizer	AdamW
Weight Decay	0.1
Beta1	0.9
Beta2	0.95
Learning Rate Schedule	Linear warmup for 25.6 million tokens up to max learning rate, then cosine decay to 0.1 $\times$ max learning rate
HMM Hidden States	512
Random Seeds	[42, 52, 62]

Table 5: Settings shared by all experiments unless otherwise specified.

Parameter	Argument
HMM Hidden States	512
Random Seeds	[42, 52, 62]
Max Learning Rate	4e-1
TinyLlama-1.1B LVD Tokens	10M
Llama2-7B LVD Tokens	5M
Llama2-7B PCA Dimension	[512, 1024, 2048, 4096]

Table 6: Experimental details for Figure 3.

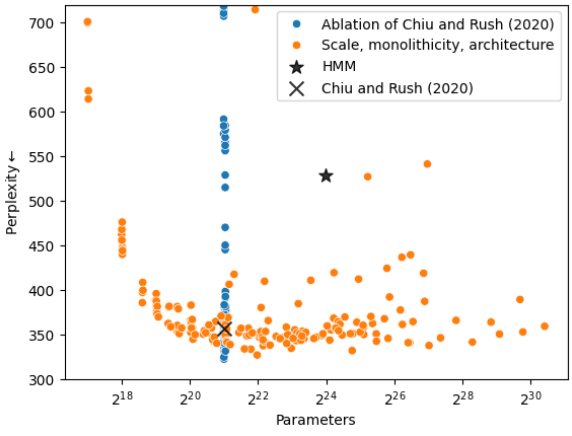

Figure 9: An overview of the experiments conducted in Section 3 involving scale, architecture (Orange) and functional form (Blue). Our baseline (★) is a randomly initialized HMM with standard parameterization and optimized with SGD.

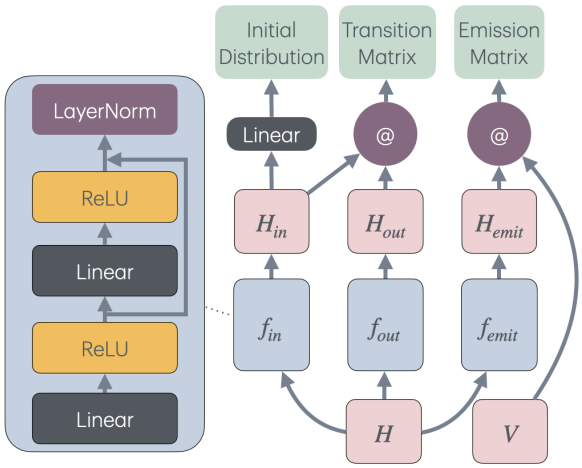

Figure 10: The neural reparameterization used by Chiu and Rush (2020). @ indicates matrix multiplication.

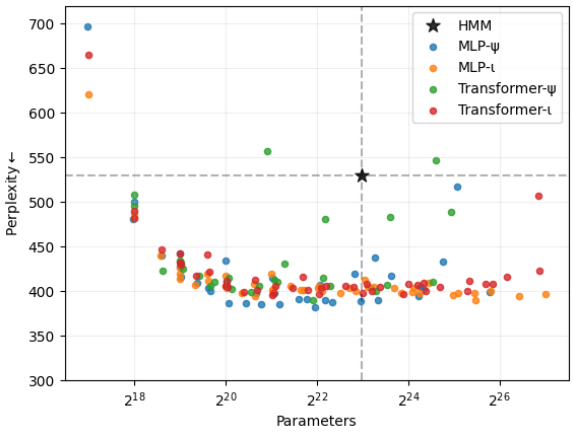

Figure 11: We repeat the experiment in Figure 2, but set hidden states to 256 instead of 512.

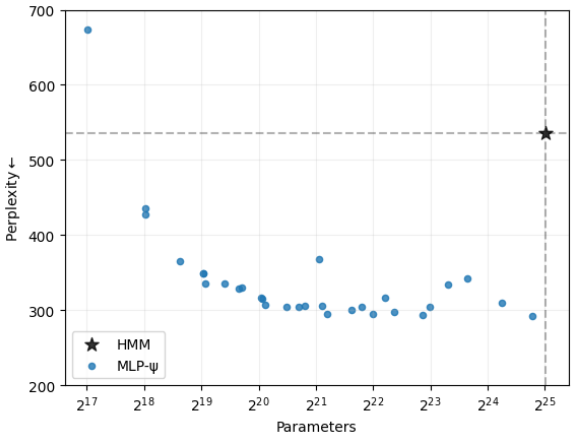

Figure 12: We repeat the experiment in Figure 2 for MLP-ψ, but set hidden states to 1024 instead of 512.

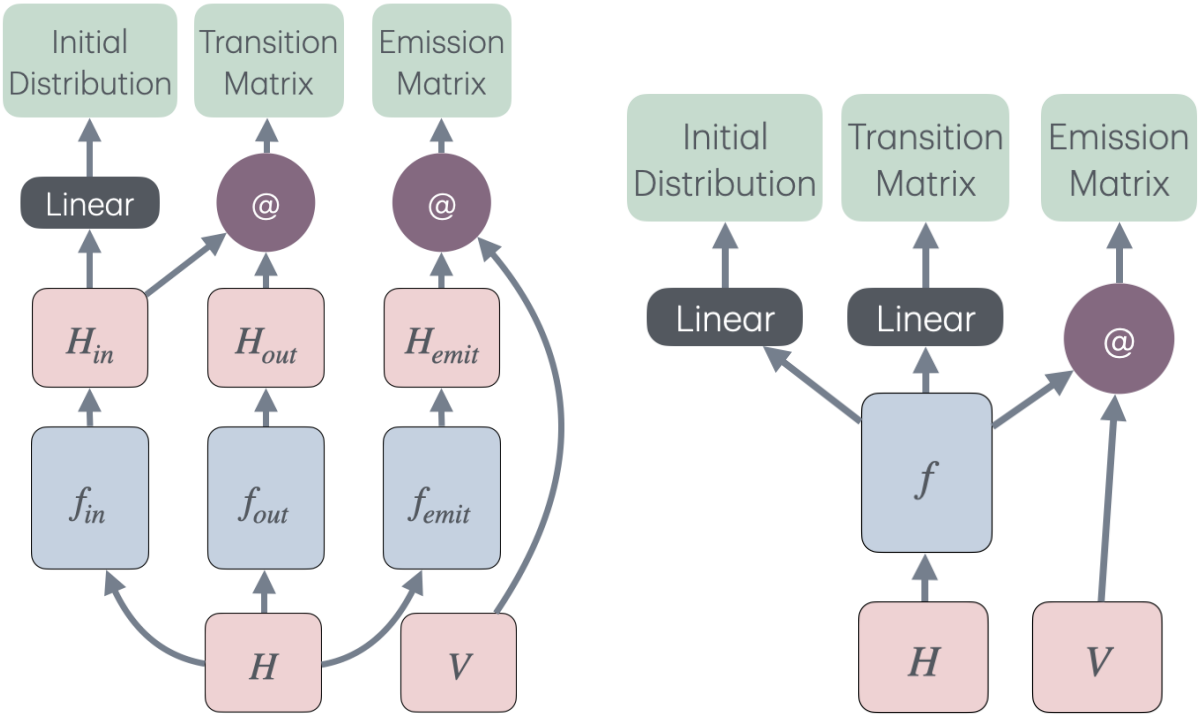

Figure 13: **Left:** Non-monolithic design (ψ). **Right:** Monolithic design (ι).

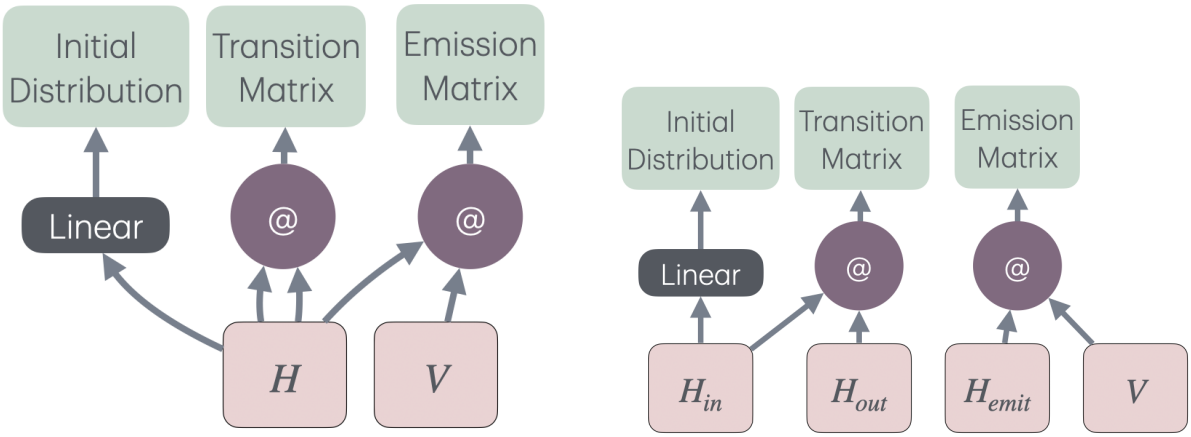

Figure 14: **Left:** Non-monolithic (ψ) design where f_i is replaced with the identity function. **Right:** Non-monolithic design (ψ) where f_i is replaced with the identity function and the hidden state embedding H is split into three independent embeddings.

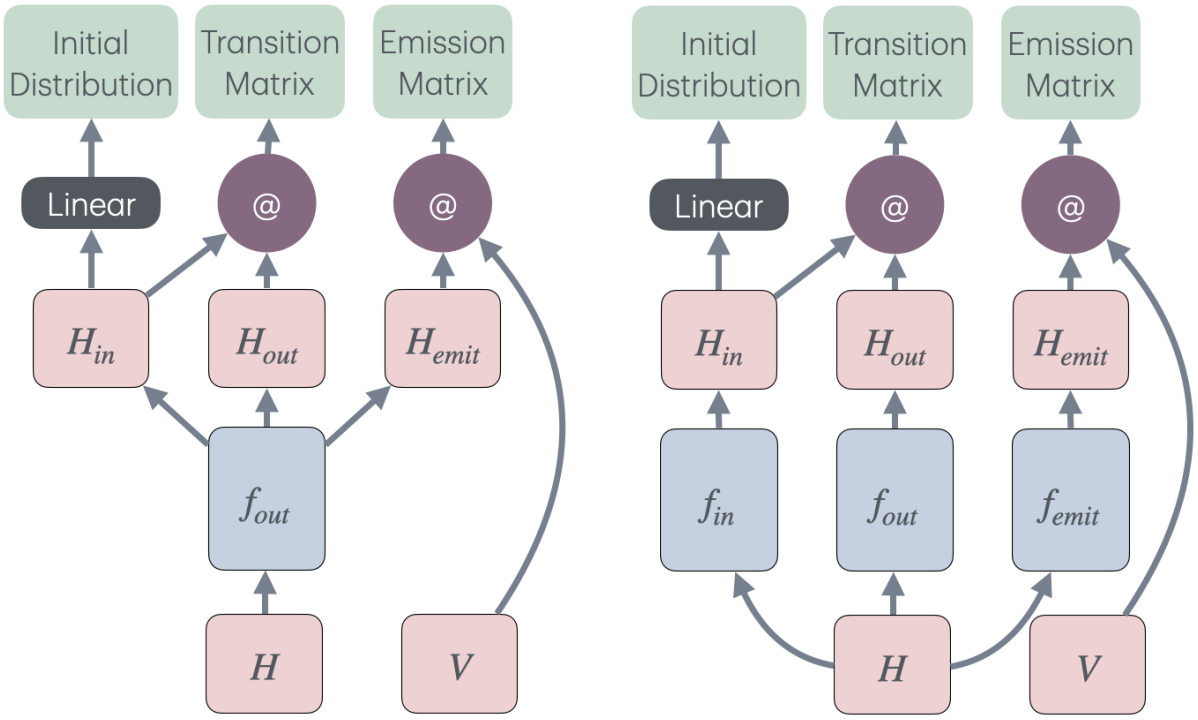

Figure 15: Examples of a monolithic architecture (left) and a non-monolithic architecture (right).

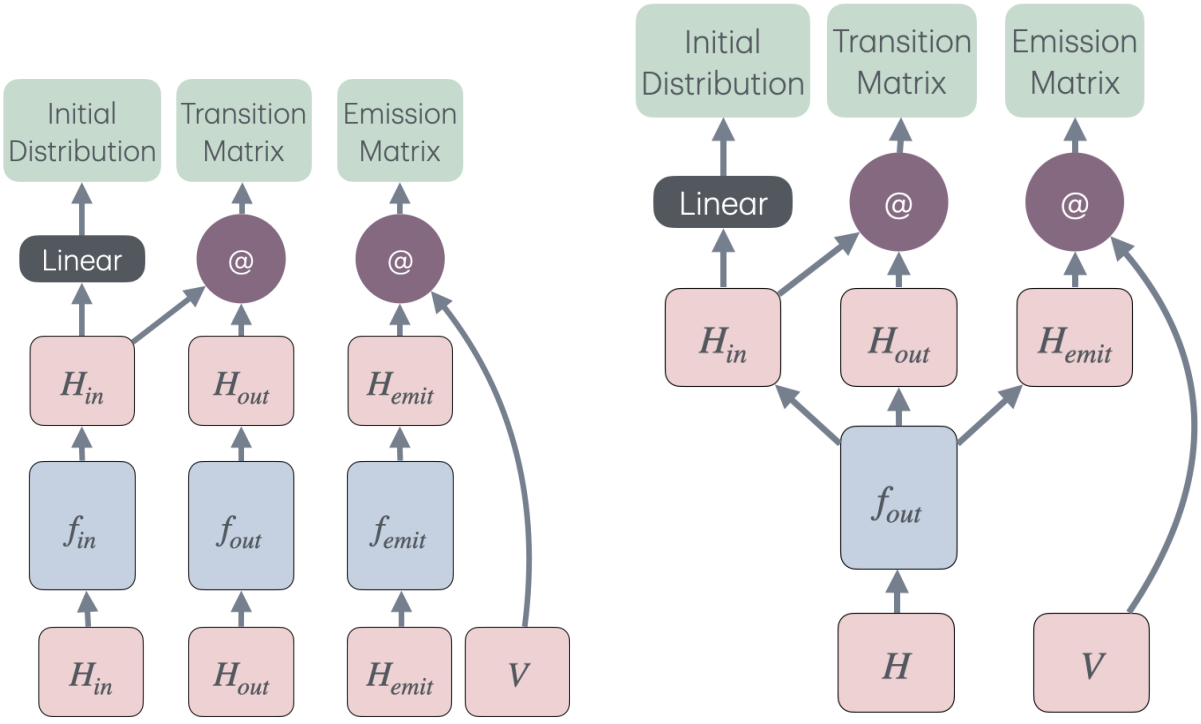

Figure 16: Examples of split hidden state embeddings (left) and merged hidden state embeddings (right).