LLM-JEPA: Large Language Models Meet Joint Embedding Predictive Architectures

Hai Huang
Atlassian
hhuang3@atlassian.com

Yann LeCun
NYU
yann.lecun@nyu.edu

Randall Balestriero
Brown University
rbalestr@brown.edu

Abstract

Large Language Model (LLM) pretraining, finetuning, and evaluation rely on input-space reconstruction and generative capabilities. Yet, it has been observed in vision that embedding-space training objectives, e.g., with Joint Embedding Predictive Architectures (JEPAs), are far superior to their input-space counterpart. That mismatch in how training is achieved between language and vision opens up a natural question: *can language training methods learn a few tricks from the vision ones?* The lack of JEPA-style LLM is a testimony of the challenge in designing such objectives for language. In this work, we propose a first step in that direction where we develop LLM-JEPA, a JEPA based solution for LLMs applicable both to finetuning and pretraining. Thus far, LLM-JEPA is able to outperform the standard LLM training objectives by a significant margin across models, all while being robust to overfiting. Those findings are observed across numerous datasets (NL-RX, GSM8K, Spider, RottenTomatoes) and various models from the Llama3, OpenELM, Gemma2 and Olmo families. Code: `https://github.com/rbalestr-lab/llm-jepa`.

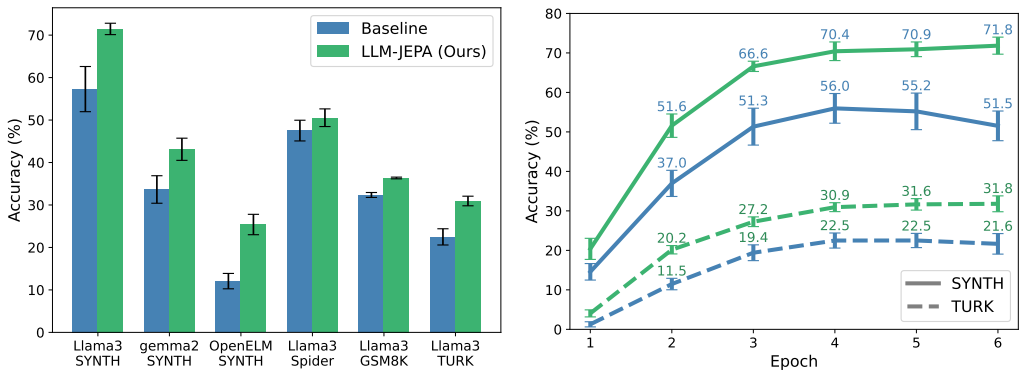

Figure 1: LLM-JEPA produces strong fine-tuned models across datasets and models.

1 Introduction

The research landscape around representation learning has been increasingly divided into two camps: (i) generative or reconstruction-based methods [6, 8, 12, 19], and (ii) reconstruction-free Joint Embedding Predictive Architectures (JEPAs) [2, 3, 4]. While the former is self-explanatory, the latter learns a representation by ensuring that different *views*, e.g., pictures of a same building at different time of day, can be predicted from each other, all while preventing a collapse of the embeddings. By moving away from input-space objectives, JEPAs training benefits from less biases [22], at the cost of potential dimensional collapse of their representation [17, 18]. That divide has been well

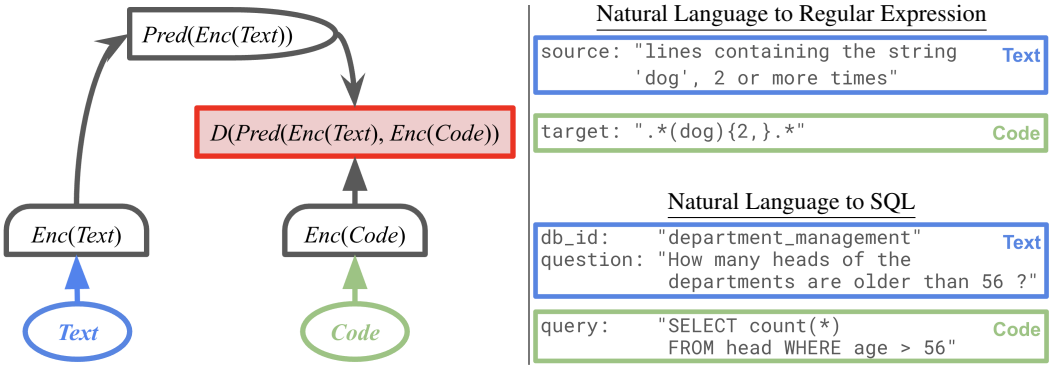

Figure 2: **Left:** JEPA applied to NLP tasks that has $Text$ and $Code$, where $Text$ and $Code$ are naturally two views of the same thing. **Right**: **(top)**: An illustration of the `NL-RX-SYNTH` dataset, where each sample consists of a description of the regular expression in natural language ($Text$) and the regular expression itself ($Code$). **(bottom)**: The `Spider` dataset, where $Text$ is the database ID and description of the SQL query and $Code$ is the SQL query itself.

studied in vision, where it was found that JEPAs offer multiple provable benefits when it comes to knowledge discovery for perception tasks. In the realm of Natural Language Processing however, reconstruction-based methods remain predominant. In fact, today's Large Language Models are mostly judged from their ability to generate samples and answers in input space in text form–making it challenging to leverage JEPA objectives.

Yet, LLMs' task also involve perception and reasoning where JEPA is known to be preferable. It thus seems crucial to adapt JEPA solutions to LLMs in the hope to showcase the same benefits as witnessed in vision. This first step is exactly what we present in this study. We propose to improve the representation quality of LLMs by leveraging a novel objective combining both the original reconstruction based loss–with an additional JEPA objective. To do so, we focus first on tasks and datasets that are inherently suited for JEPA objectives: the ones providing multiple *views* of the same underlying knowledge. One typical example is a git issue and the corresponding code diff (fig. 2) [16]. The two samples are two views–one being plain English and one being in code–of the same underlying functionality. Let's use that particular example to highlight our core contribution:

Viewing the `(text,code)` *pairs as views of the same underlying knowledge enables JEPA objectives to be utilized with LLMs, complementing the standard* `text` $\rightarrow$ `code` *generative task.*

We strongly emphasize that being able to obtain non-trivial views, such as described above, is crucial to the success of JEPA objectives. While we restrict ourselves to datasets offering those non-trivial views, developing a mechanism akin to data-augmentation in vision would enable JEPA objectives to be used on any dataset. Nonetheless, we believe that our proposed solution–coined LLM-JEPA–and empirical study will serve as a first step towards more JEPA-centric LLM pretraining and finetuning. We summarize our contributions below:

- **Novel JEPA-based training objective:** We present the first JEPA-based training objective for LLMs operating in embedding space and with different views–perfectly following vision-based JEPAs without sacrificing the generative capabilities of LLMs
- **Improved SOTA:** We empirically validate our formulation in various finetuning settings, where we obtain improvements over standard LLM finetuning solutions. We also explore pretraining scenarios showing encouraging results of LLM-JEPA
- **Extensive empirical validation:** on various model family (llama, gemma, apple/openelm, allenai/olmo), dataset (NL-RX, GSM8K, Spider, RottenTomatoes), and size.

2 JEPA-LLM: Improving LLMs' Reasoning and Generative Capabilities

The first section 2.1 provides minimal background around next-token prediction LLM objectives, used as part of the proposed LLM-JEPA loss (section 2.2). Empirical validation will then be provided in section 2.3 demonstrating clear finetuning and pretraining benefits.

2.1 Primer on Large Language Models

Contemporary LLMs are mostly built from the same core principles: stacking numerous layers of nonlinear operations and skip-connections–known as Transformers. While subtleties may differ, e.g., about positional embeddings, initialization, normalization, the main driver of performance remains the availability of high quality dataset during the pretraining stage. The training objective in itself has also been standardize throughout methods: autoregressive token-space reconstruction. Let's first denote by $\mathcal{L}_{\text{LLM}}$ the typical LLM objective used for the specific task and dataset at hand. In most cases, this will be a cross-entropy loss between the predicted tokens and the ground-truth token to reconstruction. We note that our LLM-JEPA construction is agnostic of $\mathcal{L}_{\text{LLM}}$ hence making our method general to numerous scenarios.

$$\mathcal{L}_{\text{LLM}}(\text{Text}_{1:L-1}, \text{Text}_L) = \text{XEnt}\left(\text{Classifier}\left(\text{Enc}(\text{Text}_{1:L-1})\right), \text{Text}_L\right), \tag{1}$$

where Classifier predicts the logits of the next token Text_L given the past tokens $\text{Text}_{1:L-1}$. Computation of eq. (1) is done at once over L through causal autoregression. Different stages and tasks may vary the input and output of the loss.

2.2 The LLM-JEPA Objective

Throughout this section, we will use Text and Code as concrete examples of having different views of the same underlying knowledge. It should be clear to the reader that our proposed LLM-JEPA objective handles different types of views similarly.

The construction of our LLM-JEPA objective relies on two principles. First, we must preserve the generative capabilities of LLMs and we therefore start with the $\mathcal{L}_{\text{LLM}}$ from eq. (1). Second, we aim to improve the abstraction capabilities of LLMs using the joint embedding prediction task. On top of $\mathcal{L}_{\text{LLM}}$, we then propose to add the well-established JEPA objective leading to the complete loss $\mathcal{L}$ defined as

$$\mathcal{L}_{\text{LLM-JEPA}} = \underbrace{\sum_{\ell=2}^{L} \mathcal{L}_{\text{LLM}}(\text{Text}_{1:\ell-1}, \text{Text}_\ell)}_{\text{generative capabilities (LLM)}} + \lambda \times \underbrace{d(\text{Pred}(\text{Enc}(\text{Text})), \text{Enc}(\text{Code}))}_{\text{abstraction capabilities (JEPA)}}, \tag{2}$$

where $\lambda \geq 0$ is an hyperparameter balancing the contribution of the two terms, Pred and Enc are the predictor and encoder networks respectively, and d is a metric of choice, e.g., the ℓ_2 distance. Let's now precisely describe each of those components.

The encoder. We use the `hidden_state` of the last token from the last layer as the embedding of an input sequence–as commonly done for LLM probing. Practically, we can not produce $\text{Enc}(\text{Text})$ *and* $\text{Enc}(\text{Code})$ through a single forward pass. For example, passing the concatenation of $[\text{Text}, \text{Code}]$ would require meddling with the self attention to avoid cross-view interaction which would be efficient but specific to each LLM architecture. Instead, we propose to get the encoding through two additional forward passes: one for Text, and one for Code. This incurs additional costs during training–but not during inference–see section 3 for further discussions.

The metric. When it comes comparing embeddings, it is now widely accepted in vision to leverage the cosine similarity. We thus propose to do the same for LLM-JEPA.

The predictor. We leverage the auto-regressive nature of LLM and their internal self-attention to define a *tied-weights predictor*. By introducing a special token [PRED] at the end of a given input, we allow for further nonlinear processing of the input hereby producing $Pred(\cdot)$ at the final embedding of the last layer. By reusing the internal weights of the LLM for the prediction task, we greatly reduce the training overhead and architectural design choices. Practically, we append $k \in \{0, \ldots, K\}$ *predictor tokens* to an input prompt and use the embedding of the last predictor token to be $Pred(Enc(\cdot))$. When $k = 0$, the predictor is trivial, i.e., $Pred(x) = x$.

Relation to Previous Work. Because loss functions such as $\mathcal{L}_{\text{LLM}}$ (input space reconstruction since tokens are lossless compression of the original prompts) have been shown to be sub-optimal in vision, a few LLM variations have started to employ embedding space regularizers and training objectives [5, 29]. Current solution however rely on intricate structural constraints of the embedding space, e.g., hierarchical organization and cluster, and thus fall out of the JEPA scope. We also note that our interpretation of *views* when it comes to LLM datasets, e.g., (text issue, code diff), is

Table 1: Pretraining accuracy on dataset `NL-RX-SYNTH` by Next Token Prediction ($\mathcal{L}_{\text{LLM}}$) loss vs. $\mathcal{L}_{\text{LLM}-\text{JEPA}}$ loss (our method). We inherit the best configuration from fine-tuning. Each case runs five times. Average accuracy and standard deviation are reported. We also report p-value of paired, single-tailed t-Test.

Model	Method	Accuracy (%) ↑	p-value ↓	Config
Llama-3.2-1B-Instruct	$\mathcal{L}_{\text{LLM}}$	54.38 ± 1.70	$2.94e-4$	$lr = 8e-5$
	$\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours)	60.59 ± 1.01		$\lambda = 2, k = 3$, same lr

something that has been leveraged as part of the LLM finetuning solutions–by learning to generate one from the other–without a JEPA-style loss. This includes natural language to regular expression translation [23, 30, 32], natural language to SQL parsing [11, 15, 20, 28, 31] and the more recent issue descriptions to code diffs [7, 14, 27, 33]. More intricate examples involve text-based problem solving and their counterpart program induction [1, 9, 13, 21].

2.3 Empirical Validation: LLM-JEPAs Outperforms LLMs

The JEPA loss is not implicitly minimized by $\mathcal{L}_{\text{LLM}}$. The very first observation we want to make, provided in fig. 4 lies in observing that minimizing $\mathcal{L}_{\text{LLM}}$ *does not* implicitly minimize $\mathcal{L}_{\text{JEPA}}-$ indicating that it is required to add that term during training.

LLM-JEPA Improves Finetuning. We run experiments across multiple pretrained LLMs (Llama-3.2-1B-Instruct [10], gemma-2-2b-it [26], OpenELM-1_1B-Instruct [24], and OLMo-2-0425-1B-Instruct [25]) with various datasets (NL-RX-SYNTH, NL-RX-TURK [23], GSM8K [9], Spider [31]). For a given (model,dataset) case, search for the best learning rate $lr \in \{1e-5, 2e-5, 4e-5, 8e-5\}$ based on the best possible accuracy of $\mathcal{L}_{\text{LLM}}$ after 4 epochs. Then we tune the hyperparameter specific to $\mathcal{L}_{\text{LLM}-\text{JEPA}}$, k and λ in a two dimensional grid defined by $(k, \lambda) \in \{0, 1, 2, 3, 4\} \times \{0.5, 1, 2, 4\}$ (fig. 3 and table 5). For both `NL-RX-SYNTH` and `NL-RX-TURK`, accuracy is exact match of the generated regular expression; for `GSM8K`, accuracy is exact match of the final result; and for `Spider`, accuracy is exact match of the execution result of the generated query. We provide results demonstrating how LLM-JEPA improves performances (fig. 1 left) across models (table 8), datasets (table 9), training time (fig. 1 right and fig. 5), and sizes (table 11). Examples of inputs and targets along with models' predictions and error analysis are provided in tables 6 and 7. The improved performance of LLM-JEPA holds across LoRA ranks as shown in table 3. We also provide evidence that LLM-JEPA induces an approximately linear transformation from $\text{Enc}(\text{Text})$ to $\text{Enc}(\text{Code})$ (figs. 6 and 7 and table 10).

LLM-JEPA Improves Pretraining. We pretrain Llama-3.2-1B-Instruct from randomly initialized weights on NL-RX-SYNTH dataset, a prediction is valid as long as it starts with the ground truth. We obtain that LLM-JEPA also improves the quality of the learned representation, as shown in table 1. We also conduct another pretraining experiment on `cestwc/paraphrase` containing groups of 5 paraphrases. We employ the paraphrases within a same group for the JEPA loss. Once the model is pretrained (4 epochs), we do finetuning evaluation on `rotten_tomatoes` (1 epoch). We demonstrate how JEPA pretraining improves the downstream performance post-finetuning in table 4. Note that finetuning does not employ the JEPA loss–hence showing the benefit of JEPA at pretraining stage. Lastly, we provide in table 2 generated samples demonstrating that JEPA pretraining does maintain the generative capabilities of the model when prompted with the first few tokens in the `cestwc/paraphrase` dataset.

3 Conclusion and Future Work

We introduced an alternative training objective for LLMs leveraging JEPAs. Our formulation is an exact replicate of the JEPA objective extensively used in vision–but that hadn't been adapted to language yet. Crucially, our proposed LLM-JEPA maintains the generative capabilities of LLMs while improving their abstract prompt representation as empirically validated across datasets and models. While our experiments mostly focus on finetuning, preliminary pretraining experiment are promising which we plan to scale and more thoroughly test in future work. Regarding the limitations of LLM-JEPA, the main current bottleneck is the 3-fold compute cost during training required to obtain the representations of the views. We plan to explore possible mitigation that would mask the self-attention matrix and allow for our LLM-JEPA loss to be evaluated within a single forward pass through the LLM.

References

[1] Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *arXiv preprint arXiv:1905.13319*, 2019.

[2] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15619–15629, 2023.

[3] Alexei Baevski, Wei-Ning Hsu, Qiantong Xu, Arun Babu, Jiatao Gu, and Michael Auli. Data2vec: A general framework for self-supervised learning in speech, vision and language. In *International conference on machine learning*, pages 1298–1312. PMLR, 2022.

[4] Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mahmoud Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video. *arXiv preprint arXiv:2404.08471*, 2024.

[5] Loïc Barrault, Paul-Ambroise Duquenne, Maha Elbayad, Artyom Kozhevnikov, Belen Alastruey, Pierre Andrews, Mariano Coria, Guillaume Couairon, Marta R Costa-jussà, David Dale, et al. Large concept models: Language modeling in a sentence representation space. *arXiv preprint arXiv:2412.08821*, 2024.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[7] Rocío Cabrera Lozoya, Arnaud Baumann, Antonino Sabetta, and Michele Bezzi. Commit2vec: Learning distributed representations of code changes. *SN Computer Science*, 2(3):150, 2021.

[8] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[9] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, John Schulman, Jacob Hilton, Melanie Knight, Adrian Weller, Dario Amodei, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[10] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[11] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*, 2019.

[12] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[13] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[14] Thong Hoang, Hong Jin Kang, David Lo, and Julia Lawall. Cc2vec: Distributed representations of code changes. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, pages 518–529, 2020.

[15] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. *arXiv preprint arXiv:1704.08760*, 2017.

[16] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations (ICLR)*, 2024. Oral presentation.

[17] Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding dimensional collapse in contrastive self-supervised learning. *arXiv preprint arXiv:2110.09348*, 2021.

[18] Tristan Kenneweg, Philip Kenneweg, and Barbara Hammer. Jepa for rl: Investigating joint-embedding predictive architectures for reinforcement learning. *arXiv preprint arXiv:2504.16591*, 2025.

[19] Yann LeCun. A path towards autonomous machine intelligence (version 0.9.2). *OpenReview*, 62(1):1–62, jun 2022. Version 0.9.2, released June 27, 2022.

[20] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075, 2023.

[21] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*, 2017.

[22] Etai Littwin, Omid Saremi, Madhu Advani, Vimal Thilak, Preetum Nakkiran, Chen Huang, and Joshua Susskind. How jepa avoids noisy features: The implicit bias of deep linear self distillation networks. *Advances in Neural Information Processing Systems*, 37:91300–91336, 2024.

[23] Nicholas Locascio, Karthik Narasimhan, Eduardo DeLeon, Nate Kushman, and Regina Barzilay. Neural generation of regular expressions from natural language with minimal domain knowledge. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1918–1923, 2016.

[24] Sachin Mehta, Mohammad Hossein Sekhavat, Qingqing Cao, Maxwell Horton, Yanzi Jin, Chenfan Sun, Iman Mirzadeh, Mahyar Najibi, Dmitry Belenko, Peter Zatloukal, et al. Openelm: An efficient language model family with open training and inference framework. *arXiv preprint arXiv:2404.14619*, 2024.

[25] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*, 2024.

[26] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[27] Haoye Tian, Kui Liu, Abdoul Kader Kaboré, Anil Koyuncu, Li Li, Jacques Klein, and Tegawendé F Bissyandé. Evaluating representation learning of code changes for predicting patch correctness in program repair. In *Proceedings of the 35th IEEE/ACM international conference on automated software engineering*, pages 981–992, 2020.

[28] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*, 2019.

[29] Boshi Wang and Huan Sun. Is the reversal curse a binding problem? uncovering limitations of transformers from a basic generalization failure. *arXiv preprint arXiv:2504.01928*, 2025.

[30] Xi Ye, Qiaochu Chen, Xinyu Wang, Isil Dillig, and Greg Durrett. Sketch-driven regular expression generation from natural language and examples. *Transactions of the Association for Computational Linguistics*, 8:679–694, 2020.

[31] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

[32] Zexuan Zhong, Jiaqi Guo, Wei Yang, Jian Peng, Tao Xie, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Semregex: A semantics-based approach for generating regular expressions from natural language specifications. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, 2018.

[33] Xin Zhou, Bowen Xu, DongGyun Han, Zhou Yang, Junda He, and David Lo. Ccbert: Self-supervised code change representation learning. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 182–193. IEEE, 2023.

Table 2: Generated samples by model pretrained by `paraphrase` dataset. The pretrained model is not good at terminating sentence. prompt and generation

	Ground Truth vs. Generation
Ground Truth	A garden of flowers and a bench stating "City of London."
Generation	A garden of flowers and a vase with a flower in it.....
Ground Truth	A person that is riding on a horse in a grass field.
Generation	A person that is riding in a field.....
Ground Truth	A man is riding a horse in a field.
Generation	A man is riding a horse in a field.....
Ground Truth	There are two birds standing on top of a building
Generation	There are two birds standing on a rock.....
Ground Truth	Two hawks sit on top of a roof spire.
Generation	Two hawks sit on top of a wooden bench.....
Ground Truth	.A young woman serving herself at a cookout.
Generation	.A young woman serving herself in a kitchen.....
Ground Truth	2 bowls of fruit sit on a table.
Generation	2 bowls of fruit sit on a table.....
Ground Truth	A wooden bench written 'CITY OF LONDON' at the park
Generation	A wooden bench written 'CITY and a tree.....

Table 3: Fine-tuning accuracy on dataset `NL-RX-SYNTH`, LoRA vs. full fine-tuning, both by $\mathcal{L}_{\text{LLM}}$ loss and $\mathcal{L}_{\text{LLM}-\text{JEPA}}$ loss (our method). Configuration is $lr = 2e-5, \lambda = 1, k = 1$. Each cell runs five times. Average accuracy and standard deviation are reported. At every LoRA rank, $\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours) has better accuracy. At LoRA rank 512 (22.59% trainable parameters), $\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours) achieves same accuracy as full fine-tuning, but $\mathcal{L}_{\text{LLM}}$ still has a significant gap from full fine-tuning.

LoRA Rank	**Method**	**Accuracy (%)** $\uparrow$
32	$\mathcal{L}_{\text{LLM}}$	6.09 ± 0.55
	$\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours)	7.45 ± 1.87
64	$\mathcal{L}_{\text{LLM}}$	21.09 ± 1.90
	$\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours)	32.46 ± 1.26
128	$\mathcal{L}_{\text{LLM}}$	34.21 ± 2.82
	$\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours)	48.45 ± 3.66
256	$\mathcal{L}_{\text{LLM}}$	45.57 ± 4.52
	$\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours)	60.80 ± 2.31
512	$\mathcal{L}_{\text{LLM}}$	50.18 ± 5.15
	$\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours)	72.41 ± 2.94
Full	$\mathcal{L}_{\text{LLM}}$	57.29 ± 5.32
	$\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours)	70.42 ± 2.36

A Appendix

A.1 Faster LoRA Convergence

Table 3 demonstrates that LoRA fine-tuning with $\mathcal{L}_{\text{LLM}-\text{JEPA}}$ loss not only achieves substantially higher accuracy than using $\mathcal{L}_{\text{LLM}}$ alone, but also converges more quickly. Notably, at a LoRA rank of 512, our method already reaches accuracy comparable to full fine-tuning, whereas LoRA with only $\mathcal{L}_{\text{LLM}}$ still exhibits a clear performance gap.

Table 4: Pretraining + fine-tuning `Llama-3.2-1B-Instruct` accuracy on pretraining dataset `paraphrase` and fine-tuning dataset `rotten_tomatoes` and `yelp` by Next Token Prediction ($\mathcal{L}_{\text{LLM}}$) loss vs. $\mathcal{L}_{\text{LLM-JEPA}}$ loss (our method). Note that $\mathcal{L}_{\text{LLM-JEPA}}$ is applied only at pretraining. We tune lr_{pre} and lr_{ft} by $\mathcal{L}_{\text{LLM}}$, and stick to them in LLM-JEPA pretraining. We run pretraining 5 times, and for each pretrained model, we run fine-tuning 5 times. Average accuracy and standard deviation are reported. We also report p-value of paired, single-tailed t-Test.

FT Dataset	Method	Accuracy (%) ↑	p-value ↓	Config
rotten_tomatoes	$\mathcal{L}_{\text{LLM}}$	56.57 ± 1.66	$7.38e-4$	$lr_{pre} = 8e-5, lr_{ft} = 4e-5$
	$\mathcal{L}_{\text{LLM-JEPA}}$ (ours)	57.76 ± 1.33		$\lambda = 0.5, k = 2$, same lr_{pre}, lr_{ft}
yelp	$\mathcal{L}_{\text{LLM}}$	26.46 ± 0.92	$1.00e-3$	$lr_{pre} = 8e-5, lr_{ft} = 8e-5$
	$\mathcal{L}_{\text{LLM-JEPA}}$ (ours)	27.15 ± 0.93		$\lambda = 0.5, k = 2$, same lr_{pre}, lr_{ft}

Table 5: Fine-tuning accuracy on dataset `NL-RX-SYNTH` with $\mathcal{L}_{\text{LLM-JEPA}}$ loss (ours) over various γ/λ. Configuration is $lr = 2e-5, \lambda = 1, k = 0$. We maintain $\max(\gamma, \lambda) = 1.0$ to use a fixed lr. Each cell runs five times. Average accuracy and standard deviation are reported. When $\gamma = 0.0$, it generate only empty output.

γ/λ	Config	Accuracy (%) ↑
0.0	$\gamma = 0.0, \lambda = 1.0$	0.00 ± 0.00
0.001	$\gamma = 0.01, \lambda = 1.0$	1.38 ± 0.06
0.1	$\gamma = 0.1, \lambda = 1.0$	45.80 ± 5.04
1.0	$\gamma = 1.0, \lambda = 1.0$	70.42 ± 2.36
10.0	$\gamma = 1.0, \lambda = 0.1$	67.52 ± 1.45
100.0	$\gamma = 1.0, \lambda = 0.01$	66.83 ± 3.89
∞	$\gamma = 1.0, \lambda = 0.0$	57.29 ± 5.32

A.2 Ablation Study on the Role of $\mathcal{L}_{\text{LLM}}$

One limitation of eq. (2) is that the contribution of $\mathcal{L}_{\text{LLM}}$ cannot be effectively reduced to 0. To address this, we introduce an additional hyperparameter γ to explicitly control its relative strength:

$$\mathcal{L}_{\text{LLM-JEPA}} = \underbrace{\gamma \times \sum_{\ell=2}^{L} \mathcal{L}_{\text{LLM}}(\text{Text}_{1:\ell-1}, \text{Text}_{\ell})}_{\text{generative capabilities}} + \lambda \times \underbrace{d(\text{Pred}(\text{Enc}(\text{Text})), \text{Enc}(\text{Code}))}_{\text{abstraction capabilities}}, \quad (3)$$

We vary the ratio γ/λ within $[0, 1]$ while enforcing $\max(\gamma, \lambda) = 1$ to maintain a constant learning rate. Table 5 shows that $\mathcal{L}_{\text{LLM}}$ remains essential for generative performance: when $\gamma = 0$, the fine-tuned model produces only empty outputs. This indicates that the JEPA component primarily serves as a regularization term, complementing the generative loss.

A.3 Hyperparameter Tuning for LLM-JEPA

Despite its strong accuracy gains, LLM-JEPA introduces two additional hyperparameters. As shown in fig. 3, the optimal configuration may occur at any point in the grid $(\lambda, k) \in \{0.5, 1.0, 2.0, 4.0\} \times \{0, 1, 2, 3, 4\}$, which imposes a significant cost for hyperparameter tuning. While we have not identified an efficient method to explore this space, we empirically observe that adjacent grid points often yield similar accuracy, suggesting the potential for a more efficient tuning algorithm.

A.4 Additional Generation Examples

Table 7 presents additional examples generated by fine-tuning `Llama-3.2-1B-Instruct` on the `NL-RX-SYNTH` dataset using $\mathcal{L}_{\text{LLM}}$ and $\mathcal{L}_{\text{LLM-JEPA}}$, respectively.

A.5 Overfitting Behavior in LoRA Fine-Tuning

We also conducted experiments to examine whether LoRA fine-tuning with $\mathcal{L}_{\text{LLM-JEPA}}$ exhibits similar resistance to overfitting. As shown in fig. 5, accuracy under $\mathcal{L}_{\text{LLM-JEPA}}$ generally continues

	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
$\lambda=0.5$	35.68%	36.12%	36.32%	36.13%	36.36%
$\lambda=1.0$	36.03%	33.83%	33.01%	33.86%	36.15%
$\lambda=2.0$	31.11%	33.54%	24.64%	17.27%	34.13%

(a) Llama on GSM8K, $lr = 2e-5$

	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
$\lambda=0.5$	49.43%	49.52%	49.55%	49.61%	49.18%
$\lambda=1.0$	48.13%	48.54%	49.00%	50.55%	50.13%
$\lambda=2.0$	46.95%	47.20%	48.79%	47.41%	48.50%

(b) Llama on Spider, $lr = 1e-5$

	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
$\lambda=0.5$	34.87%	34.87%	36.10%	38.57%	35.25%
$\lambda=1.0$	34.87%	35.60%	36.32%	37.28%	38.36%
$\lambda=2.0$	32.69%	34.49%	37.50%	41.70%	43.12%

(c) Gemma on SYNTH, $lr = 1e-5$

	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
$\lambda=1.0$	14.18%	10.52%	13.53%	12.96%	17.92%
$\lambda=2.0$	12.87%	12.00%	19.70%	19.33%	13.72%
$\lambda=4.0$	15.63%	13.11%	18.67%	25.40%	16.63%

(d) OpenELM on SYNTH, $lr = 8e-5$

	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
$\lambda=1.0$	87.43%	87.38%	87.30%	83.40%	83.53%
$\lambda=2.0$	87.52%	87.14%	87.33%	82.64%	78.17%
$\lambda=4.0$	87.12%	87.10%	87.30%	87.27%	87.36%

(e) OLMo on SYNTH, $lr = 8e-5$

	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
$\lambda=0.5$	57.69%	57.62%	60.26%	60.15%	60.47%
$\lambda=1.0$	57.65%	58.43%	59.92%	59.78%	59.74%
$\lambda=2.0$	57.48%	56.64%	60.03%	60.59%	60.14%

(f) Llama on SYNTH, Pretrain, $lr = 8e-5$

Figure 3: In general we didn't find any pattern on where the best accuracy could appear. It could be at either high-end or low-end of either λ or k. Furthermore, there can be dips and spikes in random locations. Nonetheless, adjacent cells have close accuracy most of times, and sweeping $(k, \lambda) \in \{0, 1, 2, 3, 4\} \times \{0.5, 1, 2, 4\}$ normally yield satisfiable results. Each cell is an average of five runs, $epoch = 4$.

Table 6: Regular expressions generated by `Llama-3.2-1B-Instruct` after fine-tuning with $\mathcal{L}_{\text{LLM}}$ loss and $\mathcal{L}_{\text{LLM-JEPA}}$ loss (ours). Color code: wrong, extra, missing

Ground Truth	$\mathcal{L}_{\text{LLM}}$	$\mathcal{L}_{\text{LLM-JEPA}}$ **(ours)**
lines not having the string 'dog' followed by a number, 3 or more times		
((dog.*[0-9].*)3,)	((dog.*[0-9].*)3,)	((dog.*[0-9].*){3,})
lines containing ending with a vowel, zero or more times		
.*(.*)(([AEIOUaeiou])*).*	(.*)(([AEIOUaeiou])*) *	(.*) (.*) { ([AEIOUaeiou])* }
lines with a number or a character before a vowel		
(([0-9])|(.)).*([AEIOUaeiou]).*	(([0-9])|(.)).*([AEIOUaeiou]).* .*	(([0-9])|(.)).*([AEIOUaeiou]).*
lines with words with the string 'dog', a letter, and a number		
((([0-9])&(dog))|([A-Za-z]))*	((([0-9])&(dog))|([A-Za-z]))*	(([0-9])&(dog))|(([A-Za-z]) *)

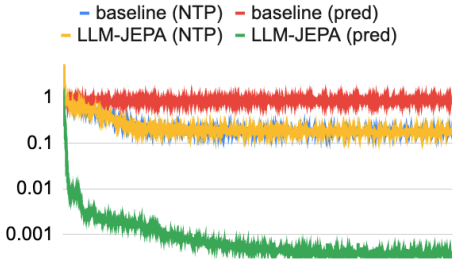

Figure 4: Losses in fine-tuning with $\mathcal{L}_{\text{LLM}}$ loss ($\mathcal{L}_{\text{LLM}}$) and $\mathcal{L}_{\text{LLM-JEPA}}$ loss ($\mathcal{L}_{\text{LLM-JEPA}}$, our method). We measure both the cross-entropy loss for next token prediction ($Loss_{LLM}$, $\mathcal{L}_{\text{LLM}}$ in chart) and JEPA prediction loss ($D(\cdot,\cdot)$, **pred** in chart), although the latter does not contribute in the baseline case. The accuracy is 51.95% for $\mathcal{L}_{\text{LLM}}$ and 71.10% for $\mathcal{L}_{\text{LLM-JEPA}}$. Since $\mathcal{L}_{\text{LLM}}$ and $\mathcal{L}_{\text{LLM-JEPA}}$ share similar $\mathcal{L}_{\text{LLM}}$ loss, the $\mathcal{L}_{\text{LLM}}$ loss cannot explain the gap between the accuracy. **pred** stays a constant in $\mathcal{L}_{\text{LLM}}$, while is minimized in $\mathcal{L}_{\text{LLM-JEPA}}$, hence **pred** should be the main reason behind the accuracy gap.

to improve with additional epochs, whereas fine-tuning with $\mathcal{L}_{\text{LLM}}$ shows clear signs of overfitting.

Table 7: More regular expressions generated by `Llama-3.2-1B-Instruct` after fine-tuning with $\mathcal{L}_{\text{LLM}}$ loss and $\mathcal{L}_{\text{LLM-JEPA}}$ loss (ours). Color code: wrong, extra, missing

Ground Truth	$\mathcal{L}_{\text{LLM}}$	$\mathcal{L}_{\text{LLM-JEPA}}$ **(ours)**
lines ending with a vowel or starting with a character		
([AEIOUaeiou].*[A-Za-z].*)+	([AEIOUaeiou].*[A-Za-z].*)+	([AEIOUaeiou].*[A-Za-z].*)+
ines containing either a lower-case letter, a vowel, or a letter		
((.*)([AEIOUaeiou]))|((.)(.*))	(.*) (([AEIOUaeiou]) |((.)(.*)))	(.*) (([AEIOUaeiou]) |((.)(.*)))
lines starting with the string 'dog' before a vowel		
(([A-Za-z])7,).*(dog).*	(([A-Za-z])7,).*(dog).* .*	(([A-Za-z])7,).*(dog).*
lines not containing a letter and the string 'dog'		
((([A-Z])+)|([a-z]))(.*)	((([A-Z])+)|([a-z]))(.*) +	((([A-Z])+)|([a-z]))(.*)
lines with a character before a vowel and the string 'dog', zero or more times		
.*(.)&([0-9])&(dog).*	.*(.)&([0-9])&(dog).* .*	.*(.)&([0-9])&(dog).* .*.*
lines with a vowel at least once before not a character		
(([A-Za-z])+).*(~([0-9])).*	(([A-Za-z])+).*(~([0-9])).* .*	(([A-Za-z])+).*(~([0-9])).*

Table 8: Fine-tuning accuracy on dataset `NL-RX-SYNTH` by Next Token Prediction ($\mathcal{L}_{\text{LLM}}$) loss vs. $\mathcal{L}_{\text{LLM-JEPA}}$ loss (our method). Each cell is the best possible accuracy over a set of configurations. Each configuration runs five times. Average accuracy and standard deviation are reported. We also report p-value of paired, single-tailed t-Test.

Model	**Method**	**Accuracy (%)** ↑	p-value ↓	**Config**
gemma-2-2b-it	$\mathcal{L}_{\text{LLM}}$	33.65 ± 3.24	$5.5e-3$	$lr = 1e-5$
	$\mathcal{L}_{\text{LLM-JEPA}}$ (ours)	43.12 ± 2.61		$\lambda = 2, k = 4$, same lr
OpenELM-1_1B-Instruct	$\mathcal{L}_{\text{LLM}}$	12.07 ± 1.81	$5.1e-4$	$lr = 8e-5$
	$\mathcal{L}_{\text{LLM-JEPA}}$ (ours)	25.40 ± 2.40		$\lambda = 4, k = 3$, same lr
OLMo-2-0425-1B-Instruct	$\mathcal{L}_{\text{LLM}}$	87.09 ± 0.36	$2.5e-3$	$lr = 8e-5$
	$\mathcal{L}_{\text{LLM-JEPA}}$ (ours)	87.52 ± 0.29		$\lambda = 2, k = 0$, same lr

Notably, the standard deviation is much higher than in full fine-tuning, likely reflecting the lower capacity of LoRA fine-tuning. An interesting pattern emerges: for $\mathcal{L}_{\text{LLM-JEPA}}$, larger standard deviations often coincide with dips in accuracy, whereas for $\mathcal{L}_{\text{LLM}}$ they tend to accompany accuracy spikes. This suggests that such fluctuations may be unreliable indicators of generalization quality.

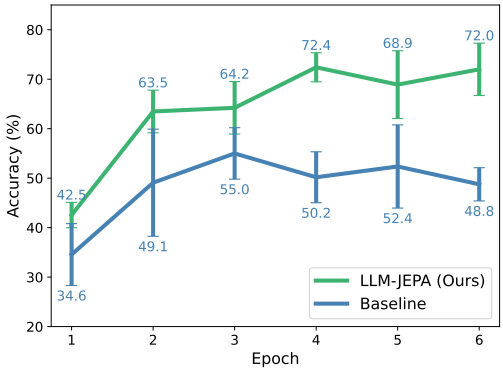

Figure 5: LLM-JEPA resists overfitting in LoRA fine-tuning. Fine-tuning with $\mathcal{L}_{\text{LLM-JEPA}}$ loss (our method) resists overfitting. When fine-tuning with $\mathcal{L}_{\text{LLM}}$ loss start to overfit, $\mathcal{L}_{\text{LLM-JEPA}}$ kept improving. However the trend is not as stable as in full fine-tuning, possibly due to limited capacity of LoRA fine-tuning.

Table 9: Fine-tuning accuracy by model `Llama-3.2-1B-Instruct`, $\mathcal{L}_{\text{LLM}}$ loss vs. $\mathcal{L}_{\text{LLM-JEPA}}$ loss (our method). Each cell is the best possible accuracy over a set of configurations. Each configuration runs five times. Average accuracy and standard deviation are reported. We also report p-value of paired, single-tailed t-Test.

Dataset	Method	Accuracy (%) ↑	p-value ↓	Config
NL-RX-SYNTH	$\mathcal{L}_{\text{LLM}}$	57.29 ± 5.32	$1.0e-3$	$lr = 2e-5$
	$\mathcal{L}_{\text{LLM-JEPA}}$ (ours)	71.46 ± 1.34		$\lambda = 1, k = 1$, same lr
NL-RX-TURK	$\mathcal{L}_{\text{LLM}}$	22.49 ± 1.91	$2.4e-4$	$lr = 2e-5$
	$\mathcal{L}_{\text{LLM-JEPA}}$ (ours)	30.94 ± 1.13		$\lambda = 1, k = 1$, same lr
GSM8K	$\mathcal{L}_{\text{LLM}}$	32.36 ± 0.58	$9.6e-5$	$lr = 2e-5$
	$\mathcal{L}_{\text{LLM-JEPA}}$ (ours)	36.36 ± 0.20		$\lambda = 0.5, k = 4$, same lr
Spider	$\mathcal{L}_{\text{LLM}}$	47.52 ± 2.44	$4.0e-3$	$lr = 4e-5$
	$\mathcal{L}_{\text{LLM-JEPA}}$ (ours)	50.55 ± 2.08		$\lambda = 1, k = 3$, same lr

Table 10: LLM-JEPA is almost a linear transformation from $\text{Enc}(\text{Text})$ to $\text{Enc}(\text{Code})$.

	$\min_X \lvert\lvert \text{Enc}(\text{Text}) \cdot X - \text{Enc}(\text{Code}) \rvert\rvert_2$	Avg. Top 100 Singular
Base model	3953.11	310.73
$\mathcal{L}_{\text{LLM}}$	3035.01	341.80
LLM-JEPA (Ours) $k = 1$	4.47	94.84
LLM-JEPA (Ours) $k = 0$	4.04	16.82

A.6 Structured Representations Induced by LLM-JEPA

We also examine the representation space to better understand how LLM-JEPA regularizes learned features. Specifically, we plot t-SNE embeddings for both Text and Code across three settings: the base model, a model fine-tuned with $\mathcal{L}_{\text{LLM}}$, and a model fine-tuned with $\mathcal{L}_{\text{LLM-JEPA}}$. As shown in fig. 6, clear structure emerges after fine-tuning with $\mathcal{L}_{\text{LLM-JEPA}}$. We hypothesize that $\mathcal{L}_{\text{LLM-JEPA}}$ enforces structure in the representation space by constraining the mapping from $\text{Enc}(\text{Text})$ to $\text{Enc}(\text{Code})$ within a narrow subspace. If this is the case, the SVD decomposition of $\text{Enc}(\text{Text}) - \text{Enc}(\text{Code})$ should yield significantly smaller singular values, which is confirmed in fig. 7. Furthermore, we hypothesize that the mapping is approximately linear. To test this, we compute the least-squares regression error, and table 10 supports this hypothesis. Together, these results suggest that LLM-JEPA promotes a near-linear transformation between Text and Code representations, which may underlie its accuracy improvements.

A.7 Performance Across Model Sizes

We also evaluate LLM-JEPA across different model sizes. As shown in table 11, we observe statistically significant improvements at all scales. Since there is no official 8B version of Llama-3.2, we instead use `Llama-3.1-8B-Instruct`, where performance collapsed due to the model's difficulty in properly terminating regular expressions. To address this, we additionally evaluate using a `startswith` criterion—that is, a prediction is considered correct if the generated regular expression begins with the ground-truth expression, removing the need for exact termination. Under this metric, we again observe statistically significant accuracy improvements.

(a) Base model: No fine-tuning

(b) Baseline: Fine-tuned by NTP loss

(c) LLM-JEPA (Ours) $k = 0$

(d) LLM-JEPA (Ours) $k = 1$

Figure 6: t-SNE plot of `Text` and `Code` representations in **(a)** Base mode without fine-tuning, **(b)** Baseline that is fine-tuned with NTP loss, **(c)** LLM-JEPA (ours) with $k = 0$, and **(d)** LLM-JEPA (ours) with $k = 1$. Clearly LLM-JEPA (ours) induced nice structure on the representations while fine-tuning with NTP loss disrupted the structure in the base model.

Table 11: Fine-tuning accuracy on `NL-RX-SYNTH` by Next Token Prediction ($\mathcal{L}_{\text{LLM}}$) loss vs. $\mathcal{L}_{\text{LLM}-\text{JEPA}}$ loss (our method). Each case runs five times. Average accuracy and standard deviation are reported. We also report p-value of paired, single-tailed t-Test. Note that Llama does not have official 3.2-8B, and we have to use 3.1-8B, which has a lower accuracy. Still LLM-JEPA sees significant improvement. We also evaluated on OLMo-2-7B.

Model	Method	Accuracy (%) ↑	p-value ↓	Config
Llama-3.2-1B-Instruct	$\mathcal{L}_{\text{LLM}}$	57.29 ± 5.32	$1.0e-3$	$lr = 2e-5$
	$\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours)	71.46 ± 1.34		$\lambda = 1, k = 1$, same lr
Llama-3.2-3B-Instruct	$\mathcal{L}_{\text{LLM}}$	74.55 ± 3.58	0.0352	$lr = 2e-5$
	$\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours)	77.16 ± 3.66		$\lambda = 2, k = 0$, same lr
Llama-3.1-8B-Instruct	$\mathcal{L}_{\text{LLM}}$	35.77 ± 6.60	0.0131	$lr = 2e-5$
	$\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours)	63.57 ± 16.81		$\lambda = 2.0, k = 0$, same lr
OLMo-2-1124-7B-Instruct	$\mathcal{L}_{\text{LLM}}$	87.26 ± 0.27	0.0345	$lr = 2e-5$
	$\mathcal{L}_{\text{LLM}-\text{JEPA}}$ (ours)	87.75 ± 0.33		$\lambda = 20, k = 2$, same lr

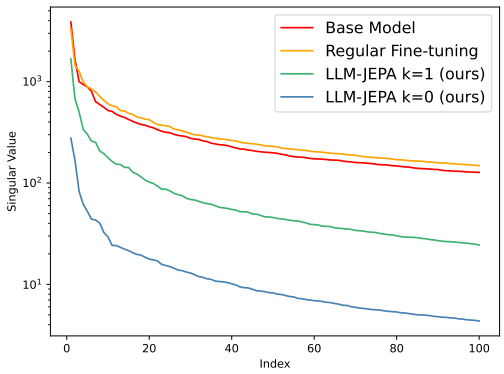

Figure 7: The top 100 singular values of $\mathrm{Enc}(\mathrm{Text}) - \mathrm{Enc}(\mathrm{Code})$. The curves of LLM-JEPA (ours) are a few magnitudes lower than that of base model and regular fine-tuning, meaning the mapping from `Text` to `Code` are confined within a narrow subspace, fostering the nice structure we see in Figure 6