Next Concept Prediction in Discrete Latent Space Leads to Stronger Language Models

ConceptLM: paper reading and evidence review

Yuliang Liu, Yunchong Song, Yixuan Wang, Kewen Ge, Alex Lamb,
Qipeng Guo, Kai Chen, Bowen Zhou, Zhouhan Lin

arXiv:2602.08984v1 | Reading: 28 September 2026

Reading presentation; the presenter is not a paper author.

Versioned paper · 20 main slides + 8 backup slides

Predict a future chunk, then generate its tokens

Research question

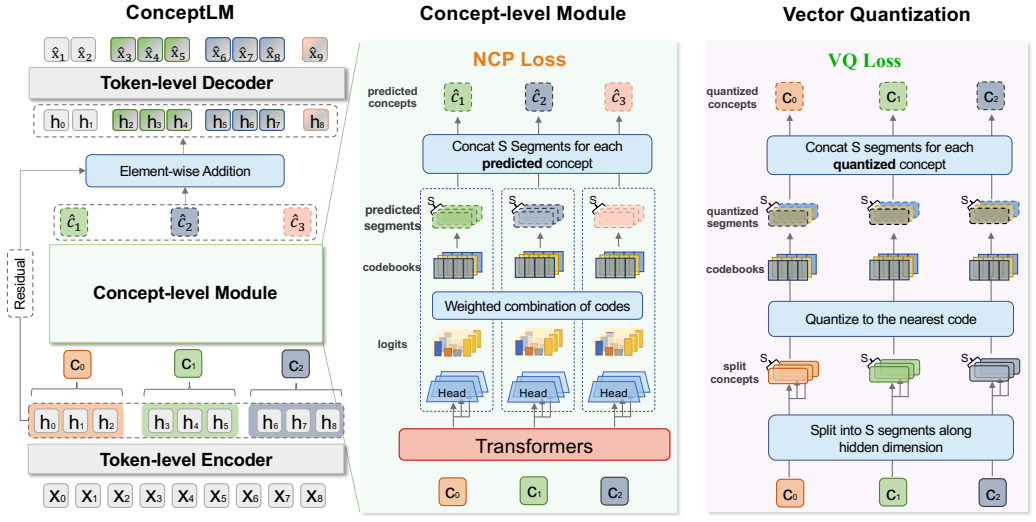
Can supervision on future multi-token representations improve a language model trained with next-token prediction?

- Pool adjacent token states into a concept; learn a segmented codebook.
- Predict the next concept and inject it into token generation.
- Evidence is strongest for small-model pretraining; the 8B gain is modest.

A concept is a learned chunk representation, not a labeled semantic unit.

Paper, Sec. 1, PDF pp. 1–2; Tables 2–3, pp. 8–9.

Two resolutions share one language model



Paper, Fig. 2, PDF p. 3. Original diagram uses 3-token groups; main experiments use $k = 4$.

Product quantization builds a compact vocabulary

$$c_j = \frac{1}{k} \sum_{r=0}^{k-1} h_{jk+r} \in \mathbb{R}^d, \qquad M = T/k$$

- Split each c_j into S feature segments of width d/S.
- Each segment has N code vectors $e_n^s \in \mathbb{R}^{d/S}$.
- N^S possible index combinations; only Nd raw code scalars.
- Main setting: $k = 4$, $N = 64$, $S =$ attention head count.

Avoiding codebook collapse

Two MLP layers with ReLU transform the codebook embeddings, inspired by SimVQ.

Paper, Secs. 2.2 and 4.1, PDF pp. 4, 6; App. E.3, p. 21. Pooling equation expanded from the text.

The predicted concept is a weighted code mixture

$$\hat{c}_j^s = \sum_{n=1}^{N} w_{j,n}^s e_n^s, \qquad \hat{c}_j = \mathrm{concat}(\hat{c}_j^1, \ldots, \hat{c}_j^S)$$

- Concept Transformers read past *continuous* states $c_{<j}$.
- S heads output N coefficients each; no N^S-way head.
- The decoder receives a continuous mixture, not a sampled code ID.
- NCP fits future continuous states with MSE, not code-ID cross-entropy.

Normalization is ambiguous in the paper; do not assume a softmax.

Paper, Secs. 2.2–3, PDF pp. 4–5; App. A, p. 18. Public Pythia code uses raw head outputs; see backup.

Causal alignment is the essential constraint

For $k = 4$, the full first chunk is available only after reading x_3.

Decoder position	Next-token targets	Added prediction
$t = 0, 1, 2$	x_1, x_2, x_3	zero
$t = 3, 4, 5, 6$	x_4, x_5, x_6, x_7	$\hat{c}_1$ from c_0
$t = 7, 8, 9, 10$	x_8, x_9, x_{10}, x_{11}	$\hat{c}_2$ from c_0, c_1

- Repeat each concept k times; delay by $k - 1$ token positions.
- The future chunk is a loss target, never a decoder input.
- Tokens are still generated autoregressively, one at a time.

Teaching example from Sec. 2.3, PDF pp. 4–5. The printed $\lfloor t/(k-1) \rfloor$ index conflicts with this rule.

Three losses support different learning roles

$$\mathcal{L} = \mathcal{L}_{\text{NTP}} + \mathcal{L}_{\text{NCP}} + \mathcal{L}_{\text{VQ}}$$

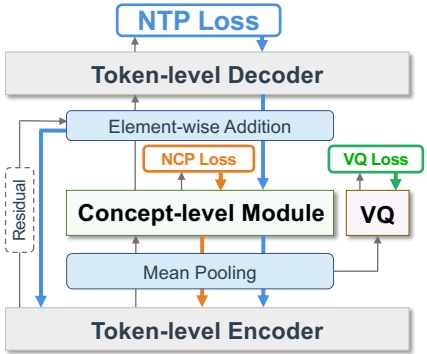
Token prediction

Cross-entropy trains token generation.

Concept prediction

$$\mathcal{L}_{\text{NCP}} = \frac{1}{M-1} \sum_{j=1}^{M-1} \|\hat{c}_j - c_j\|_2^2$$

Fit the next chunk's continuous representation.



Paper, Fig. 3 and Sec. 3, PDF pp. 3, 5–6. Public Pythia code stops gradients through the NCP target.

Codebook fitting needs an explicit gradient boundary

The paper writes the VQ objective as

$$\mathcal{L}_{\mathrm{VQ}} = \| \operatorname{sg}(c) - d_q \|_2^2 + \beta \| c - \operatorname{sg}(d_q) \|_2^2.$$

- d_q denotes the quantized vector (written d in the paper).
- The text says VQ loss updates only the quantizer.
- The second term would update c unless it is detached upstream.
- Public Pythia code detaches the pooled states before VQ, consistent with the stated intent.

A formula alone does not specify the full training graph.

Paper, Sec. 3, PDF p. 5; Fig. 3, p. 3. Code check is supplementary, not an experimental reproduction.

Three training regimes test different claims

Backbone labels	Training data	Tokens	Context
GPT-2 124M–1.5B	OpenWebText	8B	1024
Pythia 70M–410M	The Pile	300B	2048
Llama-3.1-8B	LongCtxEng	9.6B	8192

- PM adds two token layers; ConceptLM adds two concept layers.
- ContextLM tests a related hierarchy without the explicit NCP vocabulary.
- Main pretraining: 9 accuracy tasks + 4 PPL evaluations.
- Default ablations: Pythia-160M on 30B, not 300B tokens.

Paper, Sec. 4 and App. B, PDF pp. 6–9, 18–20. Sizes label the base backbone; added-layer totals differ.

GPT-2 improves over token baselines at every size

Base size	Base	PM	ContextLM	ConceptLM
124M	101.94	99.14	87.19	**84.75**
355M	52.57	52.55	43.06	**40.51**
774M	36.35	32.89	31.80	**29.61**
1.5B	29.68	30.01	**25.93**	26.38

- Reported Avg PPL $\downarrow$: arithmetic mean across four datasets.
- At 355M, in-domain PPL also improves: 17.97 $\rightarrow$ 16.81.
- At 1.5B, ContextLM wins Avg PPL and Lambada Standard PPL.

Paper, Table 1, PDF p. 7. OpenWebText, 8B tokens; main ConceptLM uses insertion before layer 1.

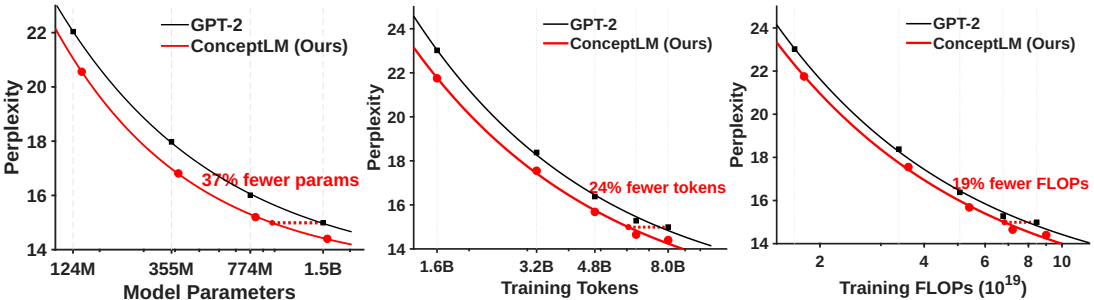
Pythia gains depend on scale and task

Base size	Base	ContextLM	ConceptLM
70M	34.6	38.2	38.2
160M	40.3	42.0	**42.9**
410M	47.5	48.2	**48.3**

- Nine-task Avg Acc (%) ↑, after 300B Pile tokens.
- At 160M: +2.6 percentage points vs Base; +0.9 vs ContextLM.
- Individual regressions remain: ARC-C 19.5 → 18.3 vs ContextLM.
- Pythia Base/ContextLM results are taken from prior work, with settings aligned by the authors.

Paper, Table 2 and Sec. 4.2, PDF pp. 7–8. No reported run-to-run uncertainty.

Scaling curves suggest separate resource savings



Near the largest GPT-2 baseline's PPL, the authors report:

- 37% fewer parameters, 24% fewer tokens, or 19% fewer FLOPs.
- These are separate comparisons, not simultaneous savings.

Paper, Fig. 1, PDF p. 1; Sec. 4.2, p. 7. Original panels; no fitted law or uncertainty intervals reported.

The 8B improvement is small and mixed

Model	MMLU	ARC-C	AGIEval	SQuAD 2.0	Avg
Trained	64.3	34.8	54.2	**36.7**	47.5
PM	64.5	35.0	53.3	36.1	47.2
ConceptLM	**64.6**	**35.1**	**54.2**	36.5	**47.6**

- All receive 9.6B tokens of continual pretraining.
- +0.4 average points vs PM; only +0.1 vs Trained.
- No confidence interval or multi-seed variance is reported.

Paper, Table 3 and Sec. 4.3, PDF p. 9. SQuAD metric variant is not specified; retain the reported score.

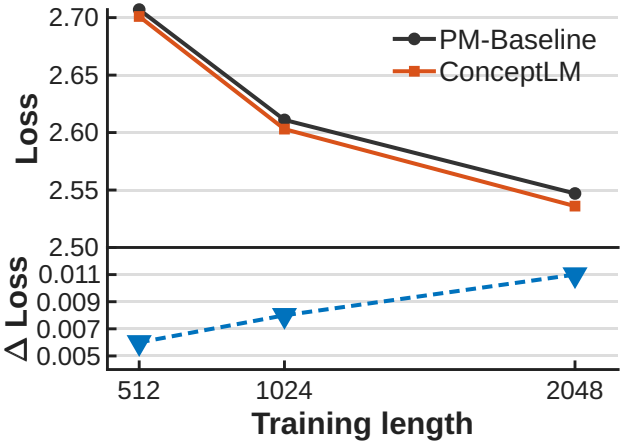
Joint objectives beat either auxiliary loss alone

Training objective	Avg Acc ↑	Avg PPL ↓
Token-model MTP	38.4	81.16
ConceptLM: NTP only	39.6	69.28
NTP + VQ	39.8	75.64
NTP + NCP	39.5	76.30
NTP + NCP + VQ	**40.3**	**68.13**

- Same ConceptLM architecture for the four objective variants.
- Full objective vs NTP-only: $+0.7$ accuracy points, -1.15 Avg PPL.
- VQ alone worsens PPL but slightly improves accuracy.

Paper, Table 4, PDF p. 10; Pythia-160M, 30B tokens. MTP is a small-model comparison only.

Longer training context increases the loss advantage



- Training lengths: 512, 1024, 2048.
- Top: language modeling loss.
- Bottom: PM minus ConceptLM loss.
- Approximate gap: 0.006 → 0.011.

No retrieval or context-extrapolation test here.

Paper, Fig. 4 and Sec. 5.3, PDF pp. 10–11. Gap values are approximate readings of the plot.

Depth results require checking the reported averages

Added layers	Token Avg PPL	Concept Avg PPL	Acc: token / concept
2	71.3500	68.1325	40.1 / 40.3
6	65.2525	54.4550	40.1 / 40.9
10	75.1350	58.3550	40.2 / 40.4

- PPL means above are recomputed from the four printed entries.
- Paper reports 59.80 for +2 token layers; its entries average 71.35.
- The concept advantage remains, but quality is not monotonic in depth.

Paper, Table 5 and Sec. 5.4, PDF p. 11. Recalculation, not a new model evaluation. Fixed-width Pythia experiment.

Cheaper concept layers reduce time relative to PM

Token layer: $O(T^2d + Td^2)$ $\longrightarrow$ Concept layer: $O(T^2d/k^2 + Td^2/k)$

Llama configuration	Training	SQuAD inference
34 token layers (PM)	80.0 h	99.0 min
32 token + 2 concept layers	71.0 h	86.0 min
Time reduction	11.25%	13.13%

- Training uses 8 H200 GPUs; theoretical FLOPs saving is about 4.69%.
- IO savings are the authors' explanation, without detailed profiling.
- The 32 token layers remain; this is not a k-fold whole-model speedup.

Paper, Table 6, PDF p. 12; App. C, pp. 19–20. Savings are against PM, not the unexpanded 32-layer model.

What the evidence does not establish

- **Semantic interpretability:** fixed token chunks need not be concepts meaningful to people.
- **A unique causal mechanism:** architecture, auxiliary supervision, and codebooks change together.
- **Reliable large-model gains:** $+0.1$ average points vs continued training has no uncertainty estimate.
- **Universal scaling:** tested depth, context length, and training budgets are limited.

Reproduction details matter

Shift indices, gradient boundaries, metric definitions, and several table averages need clarification.

Reader analysis based on Secs. 2–5 and Apps. B–E. Numerical issues do not by themselves invalidate all results.

A useful direction, with a precise claim

1. **Method:** learn a segmented concept vocabulary and predict future chunk states alongside tokens.
2. **Evidence:** generally better small-model pretraining results, with lower time than added token layers in the reported 8B setup.
3. **Next step:** isolate NCP under matched compute and repeated runs; test real long-context tasks and larger-scale training.

Discrete codebooks support continuous predictions; causal token generation remains intact.

Synthesis of the paper and this reading. Proposed next steps are reader suggestions, not reported experiments.

Sources and scope of this reading

Primary paper

Yuliang Liu et al. (2026). *Next Concept Prediction in Discrete Latent Space Leads to Stronger Language Models*. arXiv:2602.08984v1.
https://arxiv.org/abs/2602.08984v1

- Read the same-version author TeX, full main text, and Appendices A–E; checked figures and tables against the 25-page PDF.
- Original figures are reused with page locators. Recomputed values are explicitly labeled.
- Supplement: static Pythia code inspection at commit `a0ab281`; no training or benchmark reproduction.

Official repository: github.com/LUMIA-Group/ConceptLM

Accessed 28 September 2026. The following eight slides are backup material.

Backup: the main configurations are not identical

Backbone	Insert point	Batch	Learning rates
GPT-2	Before layer 1	512	10^{-3} to 4×10^{-4}
Pythia	After layer 1	1024	10^{-3} to 4×10^{-4}
Llama	After layer 1	96	10^{-5}

- Main models use two concept layers, $k = 4$, $N = 64$, $S =$ head count.
- GPT-2 also tests insertion after layer 1 in Appendix E.1.
- Paper's GPT overview says 160M; detailed tables use a 124M base.
- New Llama layers initialize from the first two base-model layers.

Paper, Secs. 4.1 and 4.3; Tables 8–12, PDF pp. 18–20. Optimizer schedules and seeds are not fully specified.

Backup: GPT accuracy improves, with a labeling caveat

Base size	Base	PM	ConceptLM L0	ConceptLM L1
124M	38.0	38.4	38.9	38.7
355M	40.6	41.4	42.3	41.5
774M	42.8	42.8	44.0	43.5
1.5B	43.9	43.5	45.1	44.7

- Nine-task Avg Acc (%); L0 is the main setting.
- L0 beats L1 here, so insertion depth affects the comparison.
- The 124M group has another row labeled "ConceptLM" at 39.1; its identity is ambiguous in Tables 2 and 14.

Paper, Table 2, PDF p. 8; Table 14, p. 23. The ambiguous row is not silently renamed as ContextLM.

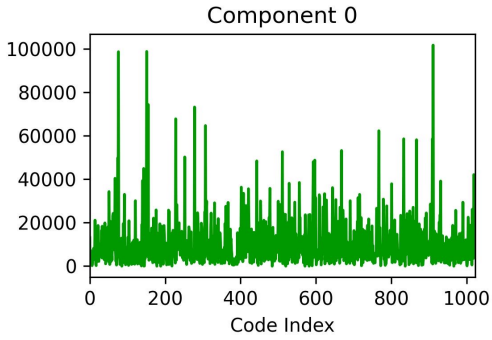
Backup: the best insertion depth depends on the metric

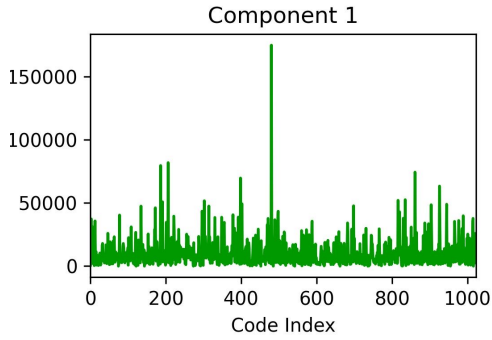
Depth	L0	L1	L3	L5	L7	L9	L11
Avg Acc	**40.5**	40.3	39.6	40.3	40.3	40.4	40.3
Avg PPL	63.25*	68.13	62.09	60.64	64.34	**55.04**	59.79

- Pythia-160M, 30B tokens; reported values shown.
- L9 is best for reported Avg PPL; L0 wins Avg Acc.
- *L0's four PPL entries average 62.5025, not 63.25.
- "Three-quarters depth is best" needs a metric and setup qualifier.

Paper, Table 15, PDF p. 24; App. B.2, pp. 18–19 and E.2, p. 21.

Backup: broad code usage is not semantic validation





Components 0 and 1 of 12; 1024 entries each.
Axes: code index and use count. Full figure is in the paper and notes.

High utilization does not imply uniform usage or interpretable concepts.

Paper, Fig. 5 and App. E.3, PDF p. 21. Two original panels cropped; $N = 1024$, not the main $N = 64$.

Backup: vocabulary size has a nonmonotonic effect

Codebook size N

N	Avg Acc	Avg PPL
8	40.0	66.88
16	40.3	**63.03**
32	40.2	69.20
64	40.3	68.13
128	**40.4**	68.06
256	40.3	70.38

Segments S, selected rows

S	Avg Acc	Avg PPL
$4H$	40.3	72.46
$2H$	**40.7**	**62.03**
H	40.3	68.13
$H/2$	40.4	62.09

H = attention head count.

More combinations do not ensure better prediction.

Paper, Tables 16–17, PDF pp. 24–25. Reported means; Pythia-160M, 30B tokens, hidden width 768.

Backup: smaller chunks trade compute for quality

Chunk size k	2	4	8	16
Avg Acc ↑	**40.7**	40.3	40.2	39.8
Avg PPL ↓	**59.80**	68.13	65.81	69.70

- $k = 2$ gives the best reported quality in this ablation.
- Larger k shortens the concept sequence and lowers its theoretical cost.
- PPL is not monotonic; $k = 8$ beats $k = 4$ on the average here.
- No measured runtime curve is supplied for these four settings.

Paper, Table 18 and App. E.5, PDF pp. 22, 25. Pythia-160M, 30B tokens.

Backup: arithmetic checks preserve the evidence boundary

- **Table 1:** Pythia-160M ConceptLM's four PPLs average 34.3175; reported Avg is 34.23. The ranking is unchanged.
- **Appendix E.4:** $64^{12} = 2^{72} \approx 4.7224 \times 10^{21}$, not 72×10^{22}. It still exceeds $50000^4 = 6.25 \times 10^{18}$.
- **Appendix C:** with $T/d = 2$, the simplified FLOPs reduction is

$$1 - \frac{390 + 64.25 \times 2}{408 + 68 \times 2} = 4.6875\%.$$

Paper, Table 1, PDF p. 7; Apps. C and E.4, pp. 20, 22. Calculations from printed values, not rerun experiments.

Backup: implementation checks clarify key ambiguities

- **VQ boundary:** Pythia detaches pooled states before quantization.
- **NCP target:** MSE uses the next pooled state with target-side detachment; predictor inputs retain gradients.
- **Causal shift:** zero-prefix, repeat by k, then drop one leading position when `shift_feature` is enabled.
- **Mixture weights:** the inspected path multiplies raw head logits by codebook vectors; no softmax before this product.

The README marks training code as pending and notes that Llama generation differs. These are static observations of public Pythia code, not claims about every original training run.

Official repository, commit a0ab281286f5c0337c35de3181cc992c562eacaa; ConceptLM_Pythiamodel.py, inspected 2026-09-28.