ENCODING PROMPTS INTO SINGLE VECTORS VIA LINEAR ACTIVATION AGGREGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models process prompts by propagating a sequence of activations through dozens of layers before generating a response. How that sequence should be encoded into a single representation that keeps the relevant information of the prompt is largely unresolved. Existing approaches sit at two extremes: heavy learned compression methods for efficiency, and simple activation-engineering heuristics like the last-token activation, chosen for convenience rather than fidelity. Motivated by the advances in mechanistic interpretability, we show that a prompt can be represented as a linear combination of its own activations. We instantiate this approach with a lightweight MLP that predicts a weight for each token. This design encodes a prompt into a single activation vector whose information can be recovered by the frozen LLM when reinjected at an early layer. Despite its simplicity, the weighted-sum representation preserves prompt information more faithfully than end-to-end compression and provides an interpretable construction. Beyond its practical implications, our analysis reveals structure in the representation space of LLMs: (i) mid-layer representations transfer meaningfully to early layers, suggesting a degree of cross-layer compatibility in how information is encoded; (ii) a single activation vector encodes a quantifiable and recoverable amount of semantic information; (iii) a weighted sum of activations provides an effective and interpretable mechanism for single-vector prompt encoding.

1 INTRODUCTION

Large language models (LLM) process prompts through dozens of layers and across entire token sequences before generating a response. Yet the task-relevant information encoded in these sequences may have a substantially more compact representation within the LLM (Hendel et al., 2023). This raises a fundamental question: can a prompt be meaningfully represented by a single vector extracted from the LLM? Such a representation would map variable-length prompts to a fixed-size vector while preserving the information needed for downstream tasks. Beyond its practical utility, such a representation could help us understand how information is encoded in LLMs and how these representations can be manipulated.

A natural way to construct such a representation is to aggregate the token activations of the prompt. In activation engineering, simple aggregation rules, such as mean pooling or selecting the last-token representation, provide a direct way to map a sequence of activations into a single vector (Subramani et al., 2022; Zou et al., 2023). Rather than fixing the aggregation rule a priori, we ask whether the model's own representations can determine how individual token activations should be combined.

Mechanistic interpretability provides several clues that the representations already formed by an LLM may be suitable building blocks for a compact prompt representation. First, work on superposition suggests that multiple features can be represented within shared activation dimensions (Elhage et al., 2022). Second, LLMs can encode higher-level information in compact representations: Todd et al. (2024) show that the behavioral effect of in-context learning can be captured by a single activation vector already present in the model. Third, a growing line of work on prompt compression has demonstrated substantial compression into a small number of learned continuous vectors (Mu et al., 2023; Ge et al., 2023), but typically relies on extensive black-box or auxiliary training rather than the LLM's existing representations. Fourth, activation engineering demonstrates that simple arith-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

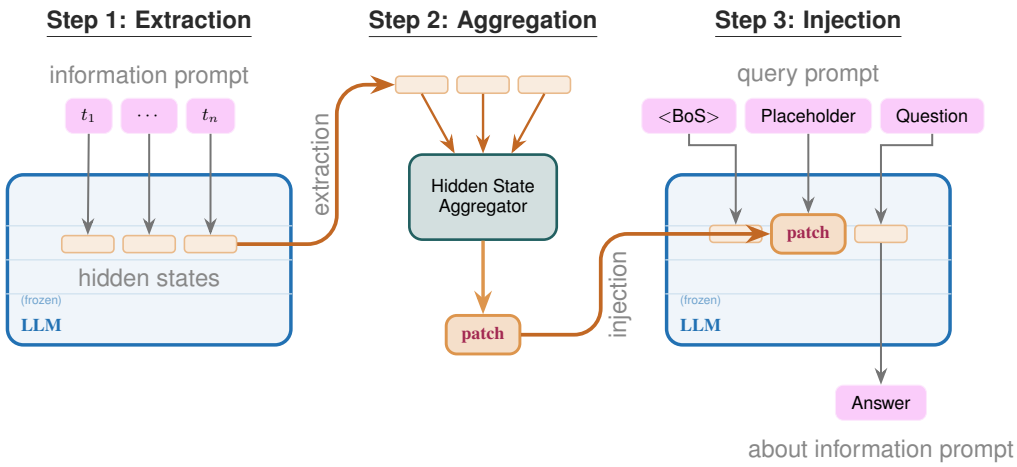


Figure 1: Overview of the framework. In the first pass, a frozen LLM extracts the hidden states of the source prompt, which the compressor aggregates into a single patch vector. In the second pass, the patch replaces the hidden state at a placeholder position, allowing the frozen LLM to answer a downstream query without direct access to the source prompt.

metic on activations can produce meaningful behavioral effects (Subramani et al., 2022), consistent with the broader linearity hypothesis (Mikolov et al., 2013; Liu et al., 2023). Taken together, these observations point toward a hypothesis that, to our knowledge, has not yet been directly tested: a prompt may be represented as a weighted sum of its own token activations. This would construct a fixed-dimensional representation directly within the LLM's existing activation space, while learning how each token activation contributes to the resulting representation.

We test this hypothesis directly with a small MLP model that predicts a scalar weight for each token activation. As illustrated in Figure 1, the method takes as input a prompt $p = [t_1, \ldots, t_T]$ and extracts the hidden-state sequence $\boldsymbol{H}^{(m)}(p) \in \mathbb{R}^{T \times d}$ at an intermediate layer m. The MLP assigns a weight to each activation, whose weighted sum produces a prompt-dependent patch vector $\boldsymbol{v} \in \mathbb{R}^d$. We evaluate whether $\boldsymbol{v}$ preserves information about the original prompt through a prompt-information retrieval task. In a second forward pass, the original prompt is replaced by a *placeholder token*, whose activation is overwritten by $\boldsymbol{v}$, followed by a query about the original prompt p. The model must then generate a response without direct access to p, relying only on the information encoded in $\boldsymbol{v}$.

We evaluate this representation hypothesis through the lens of prompt information retrieval. Because a single vector provides a limited information bottleneck, it cannot faithfully encode arbitrary prompts of any length or complexity. We therefore focus on simple knowledge-related tasks and question prompts. Despite its simplicity, the weighted-sum compressor consistently outperforms a more expressive end-to-end trained Transformer Compressor across two types of prompt datasets, while approaching the performance of the full-prompt condition. These results suggest that substantial task-relevant information can be encoded in a single vector through a simple operation over the LLM's existing activations. Based on these findings, we make the following contributions:

1. We introduce a mechanistically motivated framework for representing natural-language prompts as single activation vectors, using a lightweight linear aggregation of contextualized token activations while keeping the underlying LLM frozen.
2. We show that this simple representation preserves substantial task-relevant information, outperforming both a masked-prompt baseline and the evaluated end-to-end Transformer Compressor across task-instruction and knowledge-retrieval settings.
3. We analyze the resulting representations across layers and tokens, characterizing where prompt information is extracted and injected, how the learned weighting function distributes importance across tokens, and what information is captured by the resulting activation vectors.

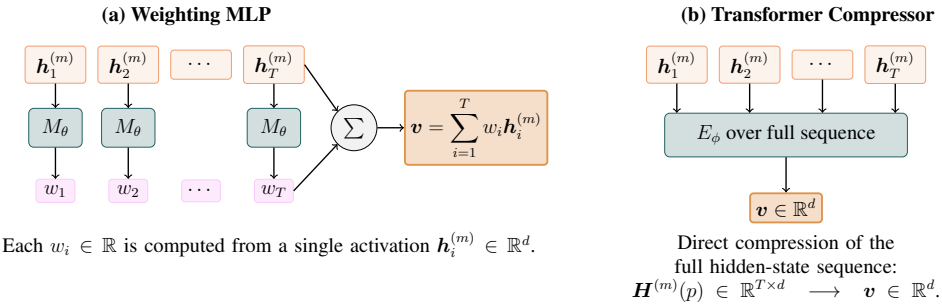


Figure 2: Comparison of our two activation aggregation methods: the Weighting MLP and the Transformer Compressor. The Weighting MLP scores each contextualized activation $\boldsymbol{h}_i^{(m)}$ separately and computes the patch vector as a weighted sum, while the Transformer Compressor consumes the full hidden-state sequence $\boldsymbol{H}^{(m)}(p)$ to directly produce a patch vector $\boldsymbol{v} \in \mathbb{R}^d$.

2 METHOD

Given a prompt $p = [t_1, \ldots, t_T]$ and an LLM with L attention layers, a hidden-state sequence

$$\boldsymbol{H}^{(m)}(p) = \left(\boldsymbol{h}_1^{(m)}, \ldots, \boldsymbol{h}_T^{(m)}\right) \in \mathbb{R}^{T\times d}$$

can be extracted at an intermediate extraction layer $m \in \{1, \ldots, L\}$. Here $\boldsymbol{h}_i^{(m)} \in \mathbb{R}^d$ denotes the hidden-state vector downstream of the token t_i at position i at layer m.

A *aggregation function* $f : \mathbb{R}^{T\times d} \to \mathbb{R}^d$ maps the sequence of hidden-states to a single *patch vector*.

$$v = f\left(\boldsymbol{H}^{(m)}(p)\right) \in \mathbb{R}^d.$$

As depicted in Figure 1, this vector is then injected at an early injection layer $e < m$, at position $k \in \{1, \ldots, T\}$, replacing the hidden-state downstream of the *placeholder* token t_k. The model then continues its forward pass from layer e onward and generates a response. This design is motivated by prior work suggesting that intermediate layers contain richer semantic representations Panickssery et al. (2023), as well as by our empirical observation that early injection performs better (see subsection 3.4). One possible explanation is that injecting the compressed representation earlier leaves more transformer layers for it to be processed and integrated before prediction.

We propose two aggregation functions f, depicted in Figure 2. The first, the *Weighting MLP* (W-MLP), operates under the strong assumption that activations can be compressed via a simple weighted sum. The second, the *Transformer Compressor* (TC), directly optimizes the aggregation of activations into the patch vector $\boldsymbol{v}$ with end-to-end training. Both methods are trained with cross-entropy between the target output of the LLM and its logits after being patched by $\boldsymbol{v} = f(\boldsymbol{H}^{(m)}(p))$. The hyperparameters of all training procedures have been optimized with grid search, details are given in Appendix F.

2.1 WEIGHTING MLP

The linear representation hypothesis suggests that the internal representations of two concepts can be summed to superimpose their effects Mikolov et al. (2013). Following these considerations, we hypothesize that the information contained in multiple activation vectors can be compressed through a weighted sum. These weights can be hand-designed, as explored in a case study presented in Appendix A. However, such hand-crafted weights are imprecise and fail to generalize beyond a specific prompt. To eliminate this manual choice, we train an MLP to predict the weights from the activations themselves. Concretely, we use a network

$$M_\theta : \mathbb{R}^d \to \mathbb{R}$$

that maps each intermediate activation $h_i^{(m)}$ to a scalar weight w_i used to form the patch vector:

$$M_\theta\left(\boldsymbol{h}_i^{(m)}\right) = w_i, \qquad \boldsymbol{v} = \sum_{i=1}^{T} w_i\, \boldsymbol{h}_i^{(m)}. \tag{1}$$

Table 1: Extraction and injection prompts for each dataset. Pink indicates the part encoded into the compressed representation $\boldsymbol{v}$. Green indicates the placeholder token where $\boldsymbol{v}$ is patched into the residual stream. Orange indicates the text generated by the LLM. Special tokens are excluded for clarity.

Dataset	**Toy Task**	**ARC-Easy**
Extraction:	`Given a country, return its capital.`	`Which of the following is a primary color?`
Injection:	`¿ Italy: Rome`	`Reply with ONLY the letter of the correct answer ¿ (A) Green (B) Orange (C) Red (D) Purple. Answer: C`

The MLP does not directly generate the full patch vector. Instead, it learns how strongly each token activation should contribute to the final aggregated representation. In our implementation, M_θ is a small feed-forward network with four layers with hidden dimensions $[2048, 1024, 512, 256]$, and ReLU activation functions.

2.2 Transformer Compressor

To explore the limits of compression, we train a transformer encoder E_ϕ that attends over the full sequence of hidden representations and produces a single vector $\boldsymbol{v}$ via a `[CLS]`-style token:

$$\begin{aligned} E_\phi &: \mathbb{R}^{T\times d} \to \mathbb{R}^d, \\ \boldsymbol{H}^{(m)} = [\boldsymbol{h}_1^{(m)}, \ldots, \boldsymbol{h}_T^{(m)}] &\mapsto \boldsymbol{v} = E_\phi(\boldsymbol{H}^{(m)})_{\texttt{CLS}}. \end{aligned} \tag{2}$$

The hidden dimension of E_ϕ is 64, with 2 attention heads and 2 layers, defined by grid-search. The TC is an end-to-end solution that does not impose any prior structure on the patch representation. While this makes the TC more expressive than the W-MLP, our experiments in Section 3.2 show that it is prone to overfitting to task-specific patterns.

3 Experiments

We evaluate whether an effective aggregation function can compress the relevant information of a prompt into a single activation-space patch vector. Our experiments are designed to answer four research questions: (1) Does the aggregated vector preserve the behavior induced by the original prompt? (2) How does performance change as the complexity of the prompt information increases? (3) How do the W-MLP and TC compare? and (4) How well do these models generalize beyond their training distribution?

3.1 Experimental Setup

Open-weight LLMs. We conduct our experiments with `Llama3.1-8B-Instruct` (Grattafiori et al., 2024), chosen for its broad availability and favorable performance-to-size trade-off. To evaluate the generality of our approach across model families and scales, we further test `Ministral3-8B-Instruct-2512` (Liu et al., 2026), `Qwen3.5-4B` (Qwen Team, 2026), and `Llama3.2-1B-Instruct`. All experiments are run on an `NVIDIA RTX A5500`.

Data. We evaluate and train our aggregation functions on two types of data: a set of homemade custom tasks and a standardized benchmark. Anonymized code and data are available here.

The **Toy Task** dataset is inspired by the task-vector experiments of Hendel et al. (2023). However, instead of representing tasks through examples, we specify them directly through instructions. We construct 11 knowledge-retrieval tasks, each defined by a simple input-output mapping (e.g., given

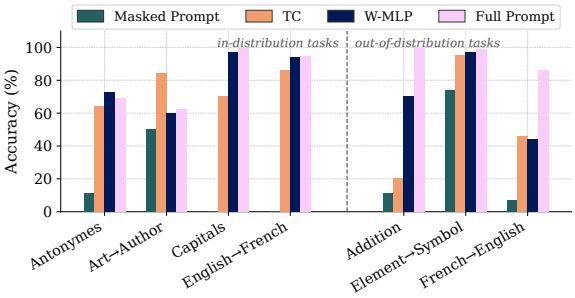


Figure 3: `Llama-3.1-8B-Instruct` test performance on a sub-sample of the *Toy Task* dataset.

Table 2: Exact-match accuracy on the *Toy Task* dataset for W-MLP, TC, and both baselines using `Llama-3.1-8B-Instruct`.

Method	Train	Test	OOD Tasks
Masked Prompt	32.55	34.03	18.45
TC	88.87	70.63	43.89
W-MLP	89.23	85.35	63.01
Full Prompt	85.75	86.92	94.95

a `country`, predict its `capital`). For each task, we create 10 prompt templates for training and 10 for evaluation, with approximately 100 (`entity`, `target`) pairs for training and 20 for evaluation per task. This dataset provides a controlled setting in which each task is precisely defined and correctness can be verified by exact string matching. It also makes it straightforward to train the aggregation functions using a cross-entropy loss over the LLM output logits. Details and examples of the dataset are provided in Appendix B.

The second dataset is **ARC-Easy** (Clark et al., 2018), a multiple-choice benchmark in which each question has four answer candidates (`A`, `B`, `C`, `D`). We compress the question itself into the representation $\boldsymbol{v}$, while the model receives only the answer choices, as depicted in Table 1. A failed compression leaves the model without access to the question, resulting in near-chance performance, whereas successful compression should recover accuracy close to the uncompressed baseline. As in the Toy Task dataset, the multiple-choice format allows direct training with cross-entropy loss over the four candidate logits. We choose ARC-Easy because its questions are relatively short and the required knowledge is largely covered by the capabilities of modern LLMs. This allows us to evaluate the aggregation strategies without prompt length or question complexity becoming the main limiting factors, unlike in more verbose and technical benchmarks such as MMLU Hendrycks et al. (2020).

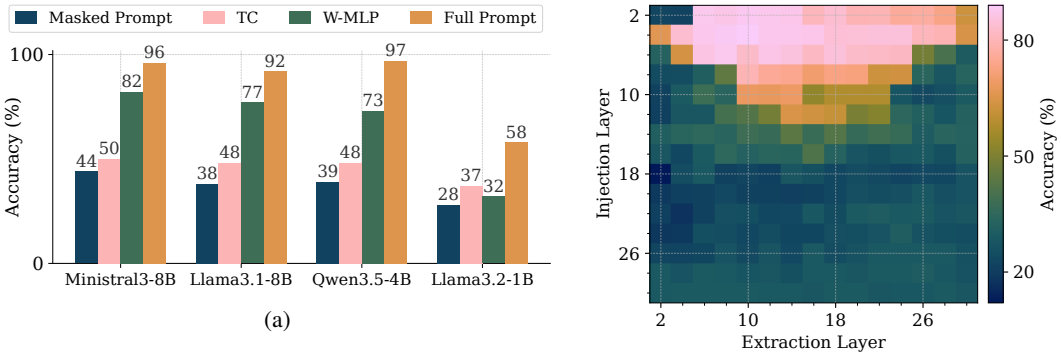
Injection Token. The patch vector $\boldsymbol{v}$ is injected by replacing the activation downstream of a designated *placeholder token* at layer e. We use the UTF-8 replacement character `U+FFFD` (rendered as ¿ here) as the placeholder, because it is unlikely to evoke semantic associations in the model's vocabulary and is available across all models. It is a single character that is consistently mapped to a single token by the tokenizer, regardless of context. We study the effect of this token choice and the injection position in Appendix C.

Baselines. We compare our compression methods against two baselines: (i) the *full-prompt baseline*, which gives the model access to the original task or question without compression and serves as an upper bound, and (ii) the *masked-prompt baseline*, which preserves the prompt structure but replaces the compressed content with the placeholder token and does not apply any patching, serving as a lower bound. Together, these baselines bracket the performance of our compressed representations. We additionally evaluate mean pooling and last-token representations as simple parameter-free aggregation baselines. Both perform below the masked-prompt condition in our setup and are therefore omitted from the main results.

3.2 Compression of Tasks Based on Instruction Prompts.

We first evaluate whether a single patch vector can preserve the behavior induced by the original prompt. We train the W-MLP and TC on eight task families from the Toy Task dataset: capitals, ISO country codes, continents, event years, English-to-French translation, art authorship, antonyms, and currency codes. We reserve three tasks for out-of-distribution evaluation: French-to-English translation, chemical element, and tuple addition. For each task family, we compare our compression methods against the two baselines defined above.

The resulting accuracies are reported in Figure 2 for `Llama-3.1-8B-Instruct`. The gap between the masked- and full-prompt baseline accuracies confirms that the task prompt contains im-



(a) (b)

Figure 4: **(a)** ARC-Easy test accuracy across LLMs, comparing the Transformer Compressor (TC), the Weighting MLP (W-MLP), and the full- and masked-prompt baselines. **(b)** Heatmap of the Weighting MLP accuracy after 1 epoch on the Toy Tasks with `Llama3.2-1B`, according to extraction layer and injection layer.

portant information, and that the entity alone is not sufficient for most tasks. We observe that both methods consistently outperform the masked-prompt baseline and recover a substantial portion of the gap to the full-prompt baseline, especially on in-distribution tasks.

We observe that the simple W-MLP method outperforms the end-to-end TC solution, suggesting that much of the task-relevant information can be manipulated through simple linear operations in representation space. On in-distribution tasks, W-MLP performs within 2% of the full-prompt baseline, suggesting that a single compressed vector can preserve most of the prompt instruction. As shown in Figure 3, its OOD performance varies substantially by task and degrades more noticeably on harder tasks.

3.3 Compression Beyond Task Prompts

ARC-Easy provides a stricter test of generalization than the out-of-distribution Toy Task setting, because each example contains a distinct one-shot prompt rather than a fixed task description. As a result, this benchmark evaluates whether the compression mechanism can generalize beyond a single prompt template while preserving information relevant for downstream multiple-choice prediction. Test accuracies across methods and LLMs are shown in Figure 4(a).

Overall, W-MLP learns a more general aggregation function and achieves test accuracy close to the full-prompt upper bound on most models. In contrast, TC appears to overfit more strongly: although it can reach training accuracy comparable to the W-MLP, its test performance drops much more sharply. On `Llama3.1-8B`, W-MLP reaches 93% training accuracy and 77% test accuracy, whereas TC reaches 87% training accuracy but only 48% test accuracy. This gap suggests again that the mechanistically motivated W-MLP induces a simpler and more stable compression function, while TC has weaker generalization.

We also find that smaller models achieve lower test accuracy with W-MLP. `Llama3.2-1B` and `Qwen3.5-4B` have hidden dimensions of $1,024$ and $2,560$, respectively, compared to $4,096$ for both `Llama3.1-8B` and `Ministral3-8B`. This suggests that smaller hidden dimensions may limit the representational capacity available for compression, and that activation-space compression may improve as the dimensionality of the model's internal representations increases.

3.4 Ablation: Extraction and Injection Layers

We next ablate the extraction and injection layers to assess their influence on compression quality. Figure 4(b) reports the training accuracy of W-MLP after one epoch across different extraction-injection layer pairs on a subset of tasks. This ablation reveals a clear trend: the best performance is obtained when activations are extracted from an intermediate layer and injected into an early layer. The strong performance of intermediate-layer extraction is consistent with prior work showing that

middle layers tend to encode more abstract and task-relevant information (Marks & Tegmark, 2023). The effectiveness of early injection may be explained by the fact that it allows more transformer blocks to process and integrate the compressed representation before prediction. The possibility to combine a middle-layer extraction with an early-layer injection is also consistent with prior work suggesting that a model's internal state remains interpretable across layers, rather than being localized to a single layer (Elhoushi et al., 2024).

The heatmap provides a coarse view of the layer-pair landscape and suggests a promising region around extraction layer 10 and injection layer 4. A finer search over this region identifies extraction at layer 12 and injection at layer 2 as the optimal configuration for `Llama3.1-8B`, which we use in the main experiments. The other models have similar optimal layers relative to their layer number.

4 What Do Patch Vectors Represent?

The results in the previous section show that a learned patch vector can recover most of the information of the original prompt. Here, we analyze what kind of information this vector captures. Specifically, we ask whether the patch vector remains close to the original prompt activations and whether the learned weighting mechanism focuses on semantically meaningful prompt tokens.

To gain intuition on the aggregation mechanism, we qualitatively inspect the learned weights assigned to each token in the weighted sum in Table 3. A consistent pattern emerges: tokens carrying dense semantic content, such as keywords or named entities, receive systematically higher weights, while low-information tokens such as determiners are comparatively suppressed.

To move beyond purely visual observations, we conduct a controlled experiment using 100 LLM-generated reformulations of the prompt *"What is the industry this company operates in?"*. Each reformulation varies in surface form while preserving task semantics, allowing us to track which token consistently receives the highest learned weight across phrasings. In 85% of cases, the highest weight is assigned to either *industry*, *field*, or *sector*, directly reflecting the target of the retrieval task. This provides quantitative support for the qualitative pattern observed in Table 3: the W-MLP identifies and prioritizes the semantically load-bearing token in the prompt. We note, however, that semantic salience is not formally defined here, so this analysis should be interpreted as a qualitative diagnostic rather than a formal measure of token importance.

In the remaining 10% of cases, the highest weight is assigned to the terminal punctuation token. Strikingly, even when it does not rank first, the interrogation mark appears as the second highest weighted token in 75% of cases across all reformulations. Rather than contradicting the semantic salience finding, this is consistent with the well-documented attention sink phenomenon in autoregressive models, whereby punctuation and sentence-boundary tokens accumulate disproportionate attention weight as aggregators of contextual information (Xiao et al., 2024; Chauhan et al., 2026). The W-MLP thus appears to exploit the same representational structure that attention heads do, further grounding the compression mechanism in known properties of transformer activation space. The focus on key semantic words and on punctuation is also consistent with the hand-crafted weights studied in Appendix A.

These observations are further corroborated by two independent metrics: the KL-divergence between the compressed patch and individual token activations, and their cosine similarity. Both metrics assign highest proximity to the same semantically salient tokens identified by the learned weights. Table 3 confirms that this interpretability extends beyond the weights themselves to the final aggregated patch vector, which remains geometrically close to the activations of task-critical tokens. Additional examples are provided in Appendix G.

5 Discussion and Future Work

Information Capacity and Accessibility. Our experiments suggest that LLMs with smaller internal representations lead to less accurate compression, independently of the LLM's overall performance. This may indicate that compression quality is constrained by the dimensionality of the hidden representation. This also aligns with the idea that sparser representations can be combined more easily, with less destructive interference, as suggested by the superposition perspective of Elhage et al. (2022). A preliminary information capacity experiment in Appendix D shows that even

Table 3: Visualization of (a) weights attributed by the learned W-MLP to each token activation; (b) Cosine-similarity between the token activations and the compressed patch vector v; (c) KL-divergence between the token activations and the compressed patch vector v. A positive metric is depicted in orange, while a negative value is depicted in blue

(a) Weight attribution of W-MLP
What is the capital of this country ?
(b) Cosine-similarity to final patch vector
Please give me the sum of the two numbers 3 and 5 .
(c) KL-Divergence to final patch vector
Given a German word respond its Chinese translation .

in the smaller `Llama3.2-1B` model, one patch vector can encode 9 task families simultaneously without a clear accuracy drop. Additionally, we note that this experiment measures not only the capacity of the vector to encode task information, but also the model's ability to recover and exploit it at inference time. Nevertheless, the question of the upper bounds on the capacity and accessibility of such representations remains open.

Interpretability by Design. Unlike post-hoc explanation methods, the learned weighting mechanism is interpretable by construction: the weights assigned by W-MLP directly expose which tokens the compression has identified as informationally relevant, without requiring any additional analysis pass. This property is noteworthy in the context of feature attribution research, where methods such as integrated gradients (Sundararajan et al., 2017) or black-box attribution approaches (Cai et al., 2025) aim to quantify the contribution of each input token to the model output. The patching weights offer a structurally similar signal but emerge naturally from the compression objective rather than being retroactively inferred. Whether these two families of scores converge empirically is an open question we leave to future work.

Limitations This work presents a mechanistic interpretability analysis of prompt representation in the activation space. We focus on the radical representation in a single vector, which is inherently lossy: arbitrarily long or information-dense sequences cannot be compressed without loss into a single vector, and we show in Appendix E that compression quality degrades as prompt length grows. Our evaluation also focuses primarily on knowledge-retrieval tasks, generalization to tasks requiring multi-step reasoning has not been established.

Future Extensions. Several directions naturally extend the current work. Most immediately, rather than compressing the full prompt into a single patch vector, one could segment the prompt into semantic blocks, each compressed into its own patch token, trading compression ratio for representational fidelity. Additionally, enforcing sparsity in individual activations prior to the weighted sum could suppress noise and render better superposition of the representations. Finally, with the growing interest in cross-layer transcoders for mechanistic interpretability (Dunefsky et al., 2024), the aggregation could be extended across layers rather than within a single one, potentially capturing richer hierarchical structure in the activations.

6 Related Work

Prompt Compression at the Token Level. Several works compress natural-language prompts into shorter token sequences. Gisting (Mu et al., 2023) learns to compress prompts into a small number of special tokens, achieving strong results but requiring full LLM fine-tuning. Auto-Compressors (Chevalier et al., 2023) and ICAE (Ge et al., 2023) similarly achieve effective long-context compression at the cost of heavy model-specific training. KV-cache and prefix caching approaches (Kwon et al., 2023; Gim et al., 2023) reduce recomputation costs, while KV-cache compression methods (Zhang et al., 2023; Xiao et al., 2023) reduce memory by retraining only selected token states. Both preserve a sequence-structured representation, rather than collapsing the prompt into a single representation.

Recent approaches push compression further towards a single representation. xRAG (Cheng et al., 2024) maps external document embeddings into the representation space of a frozen LLM, whereas 500xCompressor (Li et al., 2025) learns to compress textual context into as little as a single special token. In contrast, we investigate whether the target LLM's own contextualized activations can be compressed into a single representation through a simple and interpretable aggregation operation.

Layer-wise Information and Early Exit. Several works demonstrate that transformer layers contain sufficient information to predict the final output well before the last layer (Geva et al., 2022; Schuster et al., 2022). This suggests that representations remain in a broadly comparable, homogeneous space across depth. Early-exit methods exploit this homogeneity by halting computation once a confidence threshold is met. This body of work directly motivates our cross-layer injection strategy. To the best of our knowledge, no prior work has shown that a representation extracted at one layer can be re-injected at an earlier layer to give the model sufficient depth to process the information before generation.

Activation Steering and Representation Engineering. Prior work has shown that high-level behavioral concepts can be encoded in single vectors and injected into specific layers to redirect model behavior (Templeton et al., 2024; Ardoin et al., 2025). Representation engineering (Zou et al., 2023) and contrastive activation addition (Panickssery et al., 2023) similarly manipulate intermediate activations to steer generation. Our work is mechanistically related but distinct in objective: prior work steers *behavior*, we compress *prompts*, with the goal of faithful replication rather than redirection. Closest mechanistically to our work are task vectors (Hendel et al., 2023) and function vectors (Todd et al., 2024), which show that compact interal representations derived from in-context demonstrations can encode task- or function-level information and causally inference model behavior when reintroduced into the model. These works directly motivate our compression objective on the *Toy Task* dataset. Our work differs in what is being represented and how the representation is constructed: rather than recovering a latent task or function representation from formatted input–output demonstrations, we directly aggregate the contextualized token activations of a natural-language prompt into a single vector intended to preserve the information needed for following prompt-conditioned generation.

Superposition and Linearity in Activation Space. Elhage et al. (2022) provide theoretical grounding for the idea that LLM features are sparsely and superpositionally encoded, implying high redundancy in activation space. Liu et al. (2023) show empirically that steering vectors can be additively combined while preserving their individual effects. These findings motivate our whether a weighted sum of contextaulized token activations can retain task-relevant information from the original prompt, and whether the resulting vector lies in a region of activation space the model can interpret.

Prompt Redundancy and Information Optimality. Melamed et al. (2023) show that semantically equivalent prompts can differ radically in surface form, suggesting the model's internal task representation is not tied to specific wording. Lee et al. (2025) show that there exists an optimal token count per task, and that additional tokens do not monotonically improve performance. Our work extends this perspective to activation space, asking how much task-relevant information from a natural-language prompt can be retained in a single activation vector constructed from the model's own contextualized representations.

7 Conclusion

We have shown that relevant prompt information can be aggregated in a meaningful way with a simple learned weighted sum of the activations of an LLM. Extracting activations at a middle layer, compressing them andinjecting them at an early layer provides an effective information-compression mechanism, as confirmed by our ablations over layer depth and aggregation strategy. Beyond its practical implications, this finding offers new evidence for the linearity of LLM internal representations. Meaningful semantic content can be recovered through basic arithmetic over intermediate activations, at a scale beyond what has previously been demonstrated. We hope this opens a productive dialogue between efficiency-oriented and mechanistic interpretability research.

REFERENCES

Thibaud Ardoin, Yi Cai, and Gerhard Wunder. Where confabulation lives: Latent feature discovery in llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, August 2025.

Yi Cai, Thibaud Ardoin, and Gerhard Wunder. GEFA: A general feature attribution framework using proxy gradient estimation. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 6165–6192. PMLR, 2025.

Sonakshi Chauhan, Maheep Chaudhary, Choy Kwan Kiu, Samuel Nellessen, and Nandi Schoots. Punctuations and predicates in language models. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 5622–5636, 2026.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xrag: Extreme context compression for retrieval-augmented generation with one token. *Advances in Neural Information Processing Systems*, 37:109487–109516, 2024.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846, 2023.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable llm feature circuits. *Advances in Neural Information Processing Systems*, 37:24375–24410, 2024.

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

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*, 2023.

Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pp. 30–45, 2022.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference, 2023. URL https://arxiv.org/abs/2311.04934.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, and Archie Sravankumar. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. doi: 10.48550/ARXIV.2407.21783. URL https://arxiv.org/abs/2407.21783.

Roee Hendel, Mor Geva, and Amir Globerson. In-context learning creates task vectors. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9318–9333. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-emnlp.624.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Ayeong Lee, Ethan Che, and Tianyi Peng. How well do llms compress their own chain-of-thought? a token complexity approach, 2025. URL https://arxiv.org/abs/2503.01141.

Zongqian Li, Yixuan Su, and Nigel Collier. 500xcompressor: Generalized prompt compression for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25081–25091, 2025.

Alexander H Liu, Kartik Khandelwal, Sandeep Subramanian, Victor Jouault, Abhinav Rastogi, Adrien Sadé, Alan Jeffares, Albert Jiang, Alexandre Cahill, Alexandre Gavaudan, et al. Ministral 3. *arXiv preprint arXiv:2601.08584*, 2026.

Sheng Liu, Haotian Ye, Lei Xing, and James Zou. In-context vectors: Making in-context learning more effective and controllable through latent space steering. *arXiv preprint arXiv:2311.06668*, 2023.

Samuel Marks and Max Tegmark. The geometry of truth: Emergent linear structure in large language model representations of true/false datasets. *arXiv preprint arXiv:2310.06824*, 2023. doi: 10.48550/ARXIV.2310.06824. URL https://arxiv.org/abs/2310.06824.

Rimon Melamed, Lucas H. McCabe, Tanay Wakhare, Yejin Kim, H. Howie Huang, and Enric Boix-Adsera. Prompts have evil twins, 2023. URL https://arxiv.org/abs/2311.07064.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In Lucy Vanderwende, Hal Daumé III, and Katrin Kirchhoff (eds.), *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 746–751, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL https://aclanthology.org/N13-1090.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36:19327–19352, 2023.

Nina Panickssery, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. Steering Llama 2 via contrastive activation addition. *arXiv preprint arXiv:2312.06681*, 2023.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Nishant Subramani, Nivedita Suresh, and Matthew E Peters. Extracting latent steering vectors from pretrained language models. In *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 566–581, 2022.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pp. 3319–3328. PMLR, 2017.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L. Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. https://transformer-circuits.pub/2024/scaling-monosemanticity/, 2024.

Eric Todd, Millicent Li, Arnab Sen Sharma, Aaron Mueller, Byron Wallace, and David Bau. Function vectors in large language models. In *International conference on learning representations*, volume 2024, pp. 17282–17333, 2024.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2023. URL https://arxiv.org/abs/2309.17453.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, et al. Representation engineering: A top-down approach to AI transparency. *arXiv preprint arXiv:2310.01405*, 2023.

A Hand-Made Weighting Method

As a simple proof-of-concept baseline, we construct a patch vector directly from the prompt activations using a hand-designed weighting rule. This experiment tests whether task-relevant information can be recovered from the activation sequence of a task prompt using a simple hand-crafted heuristic without training an additional model.

Setup. We consider the country-capital task and use the base Llama-3.1-8B model in a completion-style setting. The source prompt is

```
Give me the capital city of a country I name:
```

and we extract the hidden states of all prompt tokens at an intermediate layer. At evaluation time, queries are written in the form

```
:Country:
```

For example, as `:France:`. The first colon acts as a placeholder token and is the position at which the patch is injected.

Hand-made Vector Construction. Let

$$p = (t_1, \ldots, t_T)$$

denote the tokenized source prompt, and let

$$\boldsymbol{h}_i^{(m)} \in \mathbb{R}^d$$

denote the hidden state of token t_i at the extraction layer m. In our experiment, we use extraction layer $m = 12$.

The hand-made vector consists of two components. First, we compute a position-weighted sum of all prompt activations:

$$\boldsymbol{v}_{\text{pos}} = \sum_{i=1}^{T} (0.1i) \boldsymbol{h}_i^{(m)}.$$

This gives larger weights to later tokens in the prompt. The heuristic is that later activations may contain a more complete representation of the task instruction. Second, we manually amplify a small set of task-relevant token positions. For the prompt above, we use

$$S = \{4, 5, T\},$$

which corresponds to the tokens associated with `capital`, `city`, and the final token `:`. We define

$$\boldsymbol{v}_{\text{key}} = \alpha \sum_{i \in S} \boldsymbol{h}_i^{(m)}$$

with an amplification factor $\alpha = 6$. The final patch vector is

$$\boldsymbol{v} = \boldsymbol{v}_{\text{pos}} + \boldsymbol{v}_{\text{key}},$$

which is normalized before being injected:

$$\hat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_2}.$$

Patching Procedure. For a query of the form `:Country:`, the first colon is used as the placeholder position. In the base completion model, this corresponds to token position 1, after the beginning-of-sequence token:

```
<bos> :  Country :.
```

We inject the patch vector at layer $e = 2$ by replacing the hidden state of the placeholder token:

$$\boldsymbol{h}^{(e)}_{\text{placeholder}} \leftarrow \lambda \hat{\boldsymbol{v}},$$

where λ is a scalar amplification factor. In the reproduced experiment, we use $\lambda = 100{,}000$. This is replacement patching rather than additive patching. The full configuration is summarized in Table 4.

Component	Value
Model	Llama-3.1-8B base
Prompt format	Completion-style
Source prompt	`Give me the capital city of a country I name:`
Query format	`:Country:`
Extraction layer m	12
Patching layer e	2
Selected positions S	$\{4, 5, T\}$
Amplification α	6
Scalar λ	100,000
Patch operation	Replacement
Accuracy	85.1%

Table 4: Configuration of the hand-made weighting experiment.

Results and Ablation. With the setup in Table 4, the hand-made patch vector achieves 85.1% accuracy on the country-capital training split for the base Llama-3.1-8B model.

To understand the contribution of the two terms, we also evaluate them separately. The results are shown in Table 5. The position-weighted component alone reaches 56.4%, showing that a rough positional weighting already captures some task information. The manually selected token component is stronger, reaching 80.2%. Combining both terms gives the best result.

Patch vector	Accuracy
$\boldsymbol{v}_{\text{pos}}$ only	56.4%
$\boldsymbol{v}_{\text{key}}$ only	80.2%
$\boldsymbol{v}_{\text{pos}} + v_{\text{key}}$	85.1%

Table 5: Ablation of the hand-made patch vector on the country-capital training split.

This ablation suggests that task-relevant information is not distributed uniformly across the prompt activation sequence. A small number of semantically important token activations already carry much of the useful signal. However, selecting these tokens by hand is delicate. Namely, the construction depends on the prompt wording, tokenization, patch position, and scaling factor.

This motivates the learned W-MLP aggregation model. Rather than fixing the weights w_i^{hand} manually, W-MLP learns the token weights from data and therefore can adapt the aggregation to different prompts and tasks. In particular, the hand-made construction does not transfer directly to the chat-formatted Llama-3.1-8B-Instruct model, where performance remained close to 5.9% across patching layers. We thus treat this hand-made vector as a proof of concept rather than a robust general-purpose model.

B Datasets and Prompt Formats

We use two types of datasets in our experiments: a collection of synthetic dictionary-style toy tasks and the ARC-Easy multiple-choice question answering benchmark. We now describe the input-output format, prompt format, and evaluation criterion for both settings.

Toy Task Dataset The toy task dataset consists of simple mapping tasks. Each example is a pair (x, y), where x is an input entity and y is the target string. The task prompt specifies which mapping should be applied. For example, in the country-capital task, the input is a country name and the target is its capital city.

In the main in-distribution (ID) experiments, we use eight toy tasks: capitals, ISO country codes, continents, event years, English-to-French translation, art authorship, antonyms, and currency codes. We additionally evaluate out-of-distribution (OOD) transfer on French-to-English translation, chemical element symbols, and addition of small integers. Representative examples are shown in Table 6.

Split	Task	Input x	Target y
ID	Country-capital	France	Paris
ID	ISO code	France	FR
ID	Continent	France	Europe
ID	Event year	French revolution	1789
ID	En→Fr	dog	chien
ID	Art author	Mona Lisa	Leonardo da Vinci
ID	Antonym	hot	cold
ID	Currency	Japan	yen
OOD	Fr→En	chien	dog
OOD	Element symbol	Hydrogen	H
OOD	Addition	$17 + 25$	42

Table 6: Examples of toy task input-output pairs.

For each toy task, we use a set of approximately 10 different natural-language prompts that describe the mapping to be performed. These prompts are used to extract the activation sequence from which the patch vector is constructed. Examples are given in Table 7.

Task	Example prompt
Country-capital	`Give me the capital city of a country I name`
ISO country code	`Return the ISO code of the country I name`
Continent	`Return the continent of the country I name`
Event year	`Return the year of the event I name`
English-French	`Translate the English word to French`
Art author	`Return the author of the work I name`
Antonym	`Give me the antonym of the word I provide`
Currency	`Return the currency of the country I name`

Table 7: Example prompts for the toy task dataset.

At evaluation time, the model receives the input entity x together with the patch vector. The generated output is marked correct if the target string y occurs in the generated continuation after lowercasing. This string-based criterion is used because the target may consist of more than one token.

Toy Task Prompt Format. For the instruct-model experiments, the task prompt is encoded into the patch vector, while the query contains only a placeholder and the input entity. In the notation of the main text, the compressed task prompt is the information that is replaced by the patch vector. Abstractly, the injection prompt has the form

$$\texttt{[PATCH]}\ x,$$

where x is the input entity. For example, for the country-capital task and input `Italy`, the intended behavior is

$$\texttt{[PATCH] Italy} \longrightarrow \texttt{Rome.}$$

In the actual Llama-3.1-8B-Instruct implementation, prompts are formatted using the model's chat template. The placeholder is placed in the system and the input entity is placed in the user message:

$$\begin{aligned}\texttt{system:}&\ \texttt{[PATCH]}\\ \texttt{user:}&\ x.\end{aligned}$$

After applying the chat template, the placeholder appears at a fixed token position in the system message. In our Llama-3.1-8B-Instruct, this was token position 6, and the patch vector is injected at that position.

ARC-Easy In addition to the toy task dataset, we also evaluate on ARC-Easy, the Easy split of the AI2 Reasoning Challenge Clark et al. (2018). ARC-Easy is a multiple-choice science question answering benchmark. Each example consists of a natural-language question, four answer letter candidates, and a target answer letter. In our compression setup, the question is encoded into the patch vector, while the model receives the answer choices. Thus, if compression fails, the model has access to the answer choices but not to the question. An example is given in Table 8.

Field	Example
Question	`Which object is attracted to a magnet?`
Choice A	`wooden spoon`
Choice B	`iron nail`
Choice C	`plastic cup`
Choice D	`glass bottle`
Target	`B`

Table 8: Example format for ARC-Easy.

For ARC-Easy, the extraction prompt contains the question. The injection prompt contains the placeholder and the answer choices, together with an instruction to answer with only the letter of the correct option. Abstractly, the format is

Extraction: `Question`

Injection:
```
Reply with ONLY the letter of
the correct answer
[PATCH] (A) ...  (B) ...
(C) ...  (D) ...
```

C Ablation: Placeholder Token and Patching Position

To the best of our knowledge, our specific activation-patching setting has not been previously explored, leaving several design choices open, including the choice of placeholder token and the exact position at which patching is applied. Results of the ablation study are shown in Figure 5. Our chosen configuration, patching downstream of token ¿, performs competitively and ranks among the best settings, although the margin over nearby alternatives is small. This suggests limited sensitivity to the exact placeholder token, as long as it is a neutral and consistently tokenized choice. In

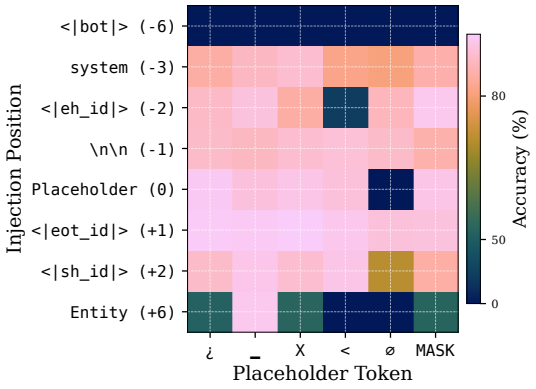


Figure 5: Heatmap of the W-MLP accuracy on the Toy Tasks as a function of the placeholder token and patching position. The y-axis indicates the token position at which the patch vector is injected, relative to the placeholder position. The absence of a placeholder token is denoted by $\varnothing$.

contrast, a clearer trend emerges along the position axis: patching further away from the placeholder token and toward either the beginning-of-text token (<|bot|>) or the assistant answer consistently degrades performance. This indicates that the intervention is most effective when applied close to the placeholder position.

These results also suggest room for improvement and motivate extensions that inject multiple patch vectors simultaneously, potentially providing richer information at the cost of a more careful positional design.

D Multi-Task Encoding in a Single Vector

To probe the information capacity of a single patched activation, we design a complementary experiment in which we directly optimize a single vector for downstream task performance, rather than a compression network. Concretely, for a set of k tasks, we train a single patch vector $\mathbf{v}$ with backpropagation such that, when injected into the model at the standard patching layer, the model successfully performs all k tasks on the given input entities. No prompt is provided at inference time: the vector alone must encode the full multi-task instruction. In order to recover the correct task, the model is prompted with the id of the task:

```
system:  [PATCH]
  user:  Task i:  [ENTITY]
```

The results are shown in Figure 6.

We train this vector for an increasing number of tasks, monitoring whether combining all tasks into a single fixed representation causes a collapse in accuracy. Surprisingly, we find no such collapse within our experimental range with up to 9 different tasks: A single vector of dimension 4096 for `Llama3.1-8B-Instruct`, and 1024 for the `Llama3.2-1B-Instruct` model, remains sufficient to encode all tested tasks simultaneously without measurable degradation. This suggests that the activation space of these models has substantially higher information capacity than a single task requires. This potentially also implies that task instructions are not competing for the same representational directions.

We note that the precise capacity limit of a single patched vector remains an open question. Establishing this limit rigorously would require a broader and more diverse task set, and we leave this to future work.

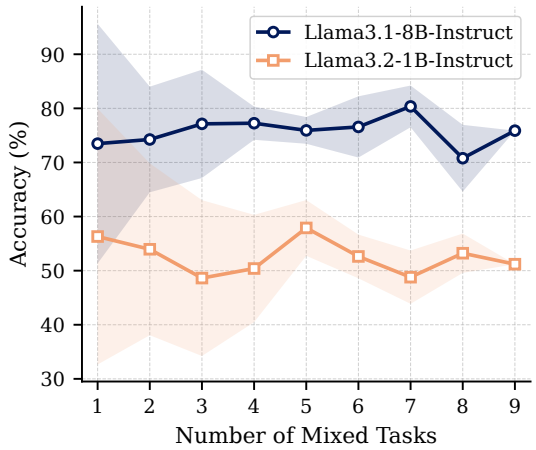


Figure 6: Multi-task encoding capacity of a single optimized patch vector. For each task group size k, we optimize one vector over a random subset of k tasks. Results are averaged over 5 runs.

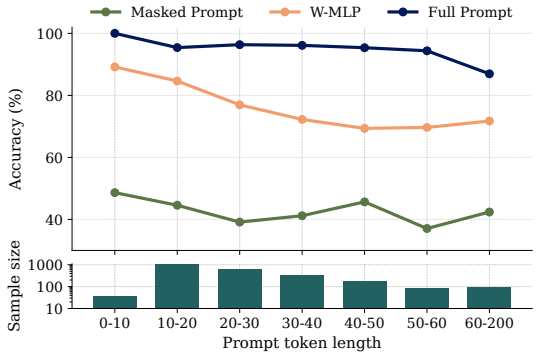


Figure 7: Accuracy of the W-MLP compression on the ARC-easy test dataset with `Ministral3-8B` according to the prompt length in number of tokens. One token holds on average 4.8 characters in this setup.

E IMPACT OF PROMPT LENGTH

Our approach intentionally explores an extreme compression regime, where the information contained in an entire prompt is represented by a single activation vector. While this design choice demonstrates the existence of task-relevant information preserved in a highly compressed representation, it also introduces an inherent limitation: arbitrarily long contexts cannot be expected to be compressed into a single vector without information loss.

As shown in Figure 7, performance gradually decreases as prompt length increases, reflecting the increasing amount of information that must be encoded into the same fixed-size representation. Nevertheless, the compressed representation remains effective, maintaining a gap of only 11 to 26 percentage points relative to the full-context baseline while using only a single token. We also observe a degradation in the full-prompt baseline for longer inputs, suggesting that part of the performance drop is due to the increased intrinsic difficulty of longer ARC-Easy examples rather than compression alone.

This experiment should therefore be interpreted as a proof of concept for activation-based prompt compression rather than an optimal compression scheme. Future improvements could explore richer representations, adaptive compression rates, or multiple activation vectors to better scale to longer contexts.

F Training and Evaluation Parameters

The training and evaluation parameters are summarized in Table 9.

Parameter	Value
Base model	Llama-3.1-8B-Instruct
Prompt format	instruction/chat template
Training tasks	8 toy task families
Extraction layer	12
Patching layer	2
Patch scaler	1.0
Learning rate	10^{-3}
Epochs	5
Batch size	W-MLP: 8; Transformer: 16
Target token position	6
Generation length	20
Temperature	0.75
Placeholder token	`U+FFFD`, rendered as ¿
W-MLP arch.	$4096 \to 2048 \to 1024 \to 512 \to 256 \to 1$
Transformer arch.	$d_{\text{model}} = 512$, 2 heads, 2 layers
Mean centering	W-MLP only

Table 9: Training and evaluation parameters.

G Examples of Weight Attribution and Similarity Measures

We provide a qualitative visualization of the learned W-MLP weights and their relation to the original prompt-token activations. These examples complement the hand-made weighting experiment in Appendix A. The hand-made construction suggested that a small number of semantically important prompt tokens can carry a large part of the useful task information. The visualizations below show that the learned W-MLP weights follow a similar pattern.

Metrics. Let

$$H^{(m)} = (h_1^{(m)}, \ldots, h_T^{(m)})$$

be the token activation sequence of a prompt p at layer m, and let v be the learned patch produced by the W-MLP.

For each token position i, we compute the cosine similarity

$$\text{sim}(v, h_i^{(m)}) = \frac{\langle v, h_i^{(m)} \rangle}{\|v\|_2 \, \|h_i^{(m)}\|_2}.$$

This measures how close the final patch vector is to each original prompt-token activation in representation space.

Next to cosine similarity, we compute a KL-divergence-based comparison between the output distribution induced by the patch vector and the output distribution induced by the corresponding prompt-token activation. Concretely, for each token activation $h_i^{(m)}$, we compare the output distribution induced by $h_i^{(m)}$ with the output distribution induced by the patch vector v:

$$D_{\text{KL}}\big(p_v(\cdot \mid x) \,\|\, p_i(\cdot \mid x)\big).$$

Lower values indicate that the token activation induces a predictive behavior closer to that of the final patch vector.

Finally, we visualize the scalar weights predicted by the W-MLP:

$$w_i = M_\theta(h_i^{(m)}), \qquad v = \sum_{i=1}^{T} w_i h_i^{(m)}.$$

These weights indicate how strongly each prompt-token activation contributes to the final compressed representation.

Qualitative Observations. Representative token-level visualizations are shown in Figures 8, 9 and 10. For each prompt, we visualize the cosine similarity to the final patch vector, the KL-divergence-based comparison, and the weights assigned by the W-MLP.

Interpretation Across the examples, the learned weights tend to assign larger values to tokens that are semantically relevant for the task, such as task words, relation words, or answer-format cues. For example, high weights often appear on tokens such as `capital`, `authored`, `animals`, or `Chinese translation`. This agrees with the intuition from the hand-made weighting experiment: task information is not distributed uniformly across the prompt, but is concentrated in a small number of informative token positions.

The cosine similarity and KL-divergence visualizations provide a complementary view. Tokens near the beginning of the prompt are generally less aligned with the final patch vector, while later and semantically more informative tokens are often closer. The capital full-prompt example also shows the effect of the prompt format: in the chat-formatted setting, the task sentence `What is the capital of this country?` appears together with special template tokens, which can also receive high similarity or weight. This suggests that the learned patch vector can capture both task content and parts of the surrounding prompt structure. These figures are intended as qualitative evidence, rather than a causal analysis of individual tokens. They provide a useful sanity check that the learned aggregation mechanism tends to focus on tokens that are meaningful for the task, rather than behaving like a uniform average over the prompt.

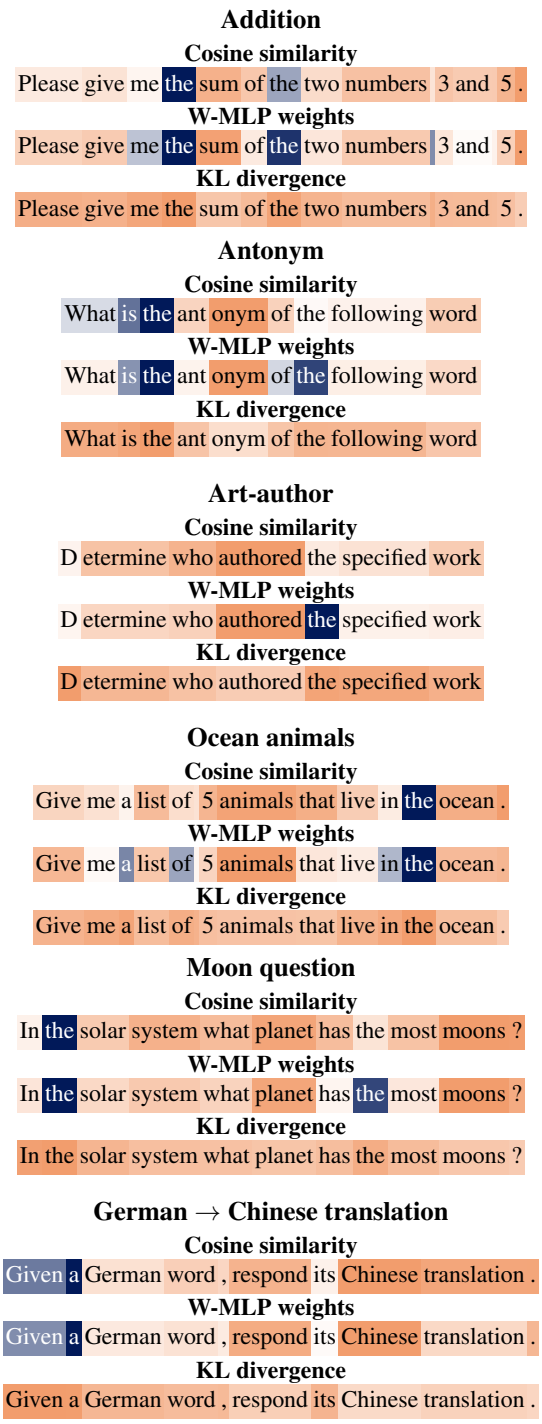


Figure 8: Qualitative token-level visualizations.

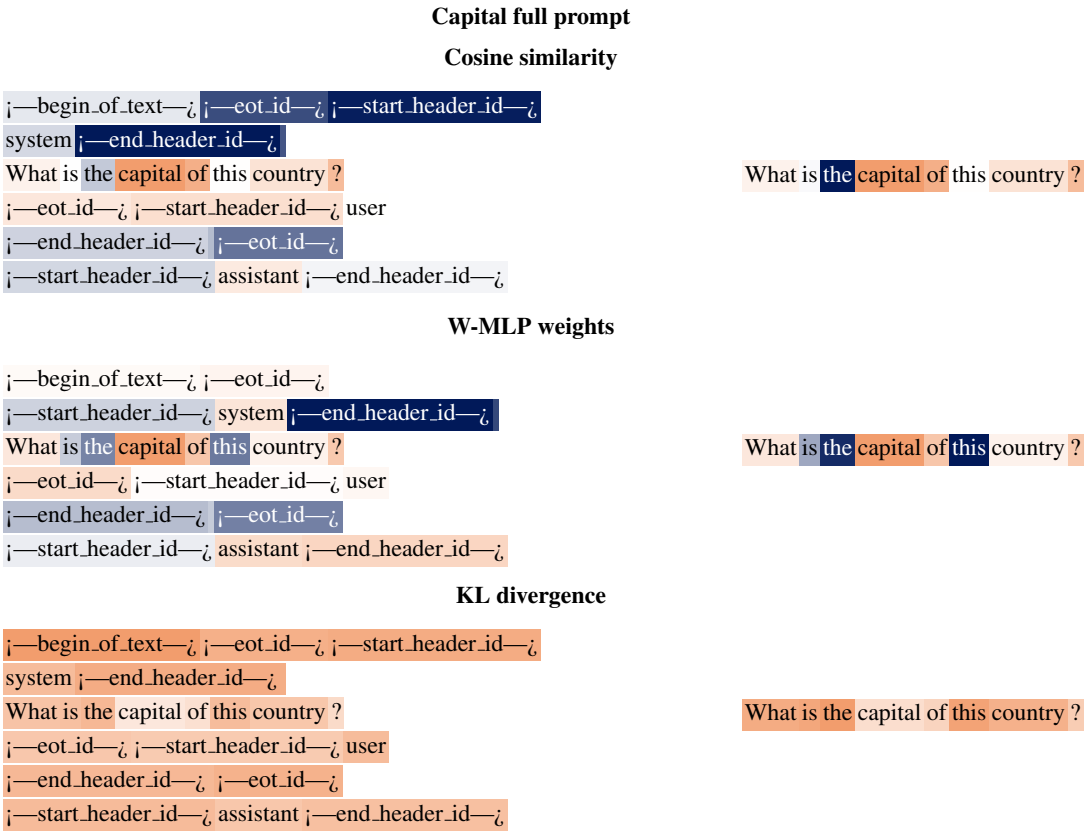


Figure 9: Comparison between the capital prompt with and without chat-template tokens. The short prompt is aligned with the natural-language task sentence inside the full prompt.

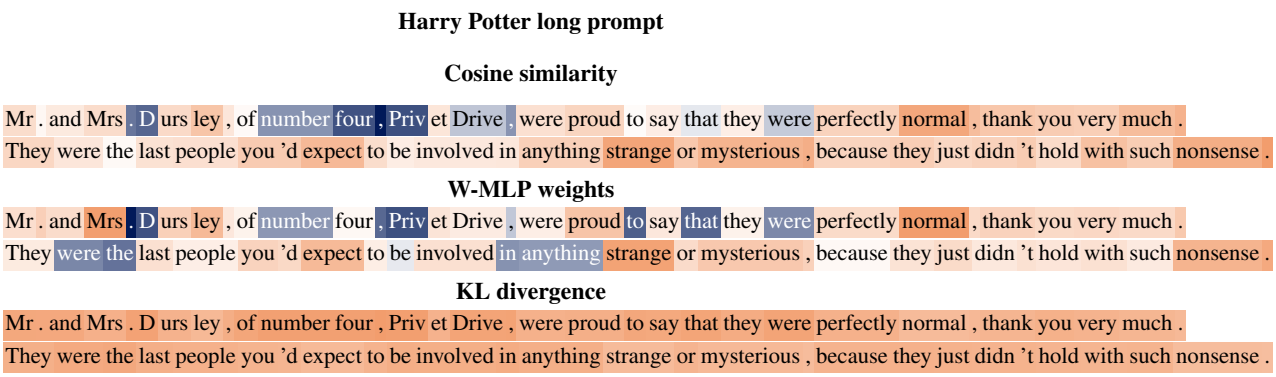


Figure 10: Qualitative token-level visualization for a longer prompt, split into two lines for readability