# The Strawberry Problem: Emergence of Character-level Understanding in Tokenized Language Models

**Adrian Cosma**[1,2], **Ștefan Ruseți**[1], **Emilian Rădoi**[1], **Mihai Dascălu**[1]
[1]National University of Science and Technology POLITEHNICA Bucharest
[2]Dalle Molle Institute for Artificial Intelligence Research (IDSIA)
{ioan_adrian.cosma, stefan.ruseti, emilian.radoi, mihai.dascalu}@upb.ro

## Abstract

Despite their remarkable progress across diverse domains, Large Language Models (LLMs) consistently fail at simple character-level tasks, such as counting letters in words, due to a fundamental limitation: tokenization. In this work, we frame this limitation as a problem of low mutual information and analyze it in terms of concept emergence. Using a suite of 19 synthetic tasks that isolate character-level reasoning in a controlled setting, we show that such capabilities emerge suddenly and only late in training. We find that percolation-based models of concept emergence explain these patterns, suggesting that learning character composition is not fundamentally different from learning commonsense knowledge. To address this bottleneck, we propose a lightweight architectural modification that significantly improves character-level reasoning while preserving the inductive advantages of subword models. Together, our results bridge low-level perceptual gaps in tokenized LMs and provide a principled framework for understanding and mitigating their structural blind spots. We make our code publicly available.

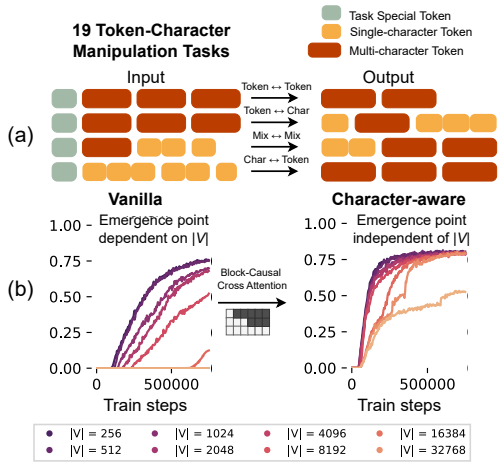


Figure 1: Tokenization obscures character composition of words. (a) We develop 19 token manipulation tasks spanning token↔token, token↔char, char↔token, and mixed settings, isolating character reasoning from semantics. (b) In a standard transformer decoder LM, character-level understanding emerges abruptly and late in training, having the emergence point pushed back as the tokenizer vocabulary size $|V|$ increases; adding our character-aware module brings early emergence that is effectively independent of $|V|$.

## 1 Introduction

LLMs have exhibited impressive capabilities in solving olympiad math problems (Trinh et al., 2024), playing open-world games (Wang et al., 2023) and passing bar exams (Achiam et al., 2023). However, paradoxically, LLMs often struggle in simple tasks that involve character-level reasoning and manipulation[1]. A growing body of work shows that language models are brittle to misspellings, struggle with character-level tasks (Shin and Kaneko, 2024; Zhang and He, 2024), and fail even simple reasoning tasks that require access to words' constituent letters. One such infamous problem, the so-called "Strawberry Problem", consists of counting the number of "r"s in the word "strawberry", a problem that many foundational models struggle to consistently answer even today (Chai et al., 2024).

The root of this problem lies in text tokenization. Tokenization is heavily used in modern language models (Sennrich et al., 2015), in which the raw text is compressed into sequences of multi-character subword tokens. This comes at a cost:

[1]This can be seen as a form of Moravec's Paradox (Newell, 1983) - reasoning is easy; perception is hard.



*Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 28252–28263
November 4-9, 2025 

tokenization severs the connection between words and their characters, limiting the model's reasoning capabilities about characters and morphology. Paradoxically, while tokenization imposes a structural bottleneck, it also provides critical inductive biases (Rajaraman et al., 2024), and cannot be completely avoided. In the absence of tokenization, models trained directly on characters or bytes (e.g., Xue et al. (2022); Wang et al. (2024)) learn more slowly and require more data to generalize. Thus, there is a fundamental tension: tokenization improves efficiency and generalization at the cost of losing fine-grained perceptual access to the underlying text (Chai et al., 2024).

In this work, we argue that the emergence[2] of character-level understanding is best modeled through the lens of mutual information and theories of concept emergence (Lubana et al., 2025). Our thesis is that learning character composition of words is equivalent to learning underreported commonsense facts (Do et al., 2024). Human-written text almost never directly mentions the characters inside words, as this is self-evident for humans upon reading a text[3].

To better understand this phenomenon, we construct a suite of 19 synthetic tasks that require models to reason about the character composition of tokens in a strictly controlled, synthetic, setting (see Figure 1). We show that performance on these tasks emerges late in training, is modulated by vocabulary size and composition, and aligns with theoretical predictions from concept percolation models of emergence (Lubana et al., 2025). A model trained on tokenized sequences gets little signal about characters and must slowly reconstruct this mapping across many training steps, and we show and that real world tokenizers worsen this effect.

We introduce a simple architectural intervention to address this bottleneck: a block cross-attention mechanism that exposes character-level information to the model alongside token embeddings. Unlike existing byte-level or hybrid models (Tay et al., 2022; Neitemeier et al., 2025), our approach preserves the inductive benefits of subword tokenization while mitigating its perceptual blindness. We show that this design improves character-level reasoning with minimal additional cost, and that it effectively raises the mutual information between tokens and characters during training. Our experiments are reproducible and we make our code publicly available[4].

Our work makes the following contributions:

1. We develop a benchmark of 19 synthetic tasks to train and evaluate character-level understanding in tokenized LMs, revealing slow and sudden emergence patterns across training. We show that character learning is slow and dependent on vocabulary size and number of characters per token, even in an idealized setting, with the effect being heightened using real-world data.

2. We find that we can explain the emergence of character-level understanding through concept percolation theories of emergence (Lubana et al., 2025). Our results hint that learning character-token correspondences is not fundamentally different from learning abstract concept-property associations.

3. We propose a lightweight character-aware architecture that increases the empirical mutual information between tokens and their characters. Our design adds a small cross-attention module that allows each token to attend to its constituent characters, while still using tokens as inputs and outputs. We validate our architecture on models using pretrained tokenizers and on real-world data, indicating significant improvements in character-level tasks compared to a standard language model operating on tokenized sequences.

## 2 Related Work

**Internal Character Representation in Language Models.** Recent works (Shin and Kaneko, 2024; Zhang and He, 2024; Chai et al., 2024) highlight the limitations of Language Models in understanding the character-level structure of tokens. Shin

[2]We consider the definition by Anderson (1972): *"Emergence is when quantitative changes in a system result in qualitative changes in behavior"*. This applies not only to model size, but, for example, to increasing training duration.

[3]A phenomenon that can be understood as a form of non-reporting bias (Gordon and Van Durme, 2013; Shwartz and Choi, 2020), in which the rare and the interesting are overrepresented at the expense of the trivial.

[4]https://github.com/cosmaadrian/strawberry-problem

and Kaneko (2024) argued that LLMs lack robust internal representations of the character composition of tokens. In discussing future directions, they propose embedding tokens with character information and positional encodings - an approach closely aligned with our method. Zhang and He (2024) evaluated LLMs on 15 simple text-editing tasks and found that models struggle without fine-tuning; the authors found that supervised fine-tuning for text editing substantially improved performance without harming general capabilities. Kaplan et al. (2025) argued that LLMs implicitly combine subword units into full words and exploited this finding to improve efficiency by adding dedicated word-level tokens. In contrast, our focus is the reverse: we investigate how LLMs can be encouraged to decompose tokens into their constituent characters. Different from these previous works (Shin and Kaneko, 2024; Zhang and He, 2024), we isolate and analyze character-level capabilities in a tightly controlled synthetic setting, revealing their emergence dynamics during training.

**Character-aware models.** There have been multiple works attempting to design character-aware models, such as the works of Tay et al. (2022); Islam et al. (2022); Wang et al. (2024) *et alia*, by operating directly on characters, bypassing the need for tokenization. One downside of such models is that they model directly the input and output characters, resulting in long generation sequences and decreased efficiency (Rajaraman et al., 2024). In contrast, our model operates directly on multi-character tokens, utilizing the inductive bias given by tokenization, while incorporating character information for each token.

The construction of neural architectures with a hierarchical structure of representations has been a common design pattern, usually in domains that require either long contexts (He et al., 2024; Nawrot et al., 2021; Wu et al., 2021) or high-detail granularity (Chen et al., 2021). In contrast to previous works, we design our model for causal next-token prediction in mind, and not for MLM or for computing general representations.

**Theories of capability emergence.** Our work is also related to recent theoretical perspectives on capability emergence in models (McKenzie et al., 2023; Hupkes et al., 2020; Lubana et al., 2025; Park et al., 2024). Hupkes et al. (2020) designed controlled tasks to test models' capability to compositionally generalize. While they operated at the level of tokens, we explore whether similar emergent behaviours arise at the lower level of token decomposition into characters. McKenzie et al. (2023) identified tasks in which larger LLMs perform worse than their smaller counterparts. Among these tasks is the "resisting correction" task, in which the model automatically, but wrongly, corrects a misspelled token. However, the study's focus was on model scale and compute allocation, while we explore the relationship between vocabulary size and performance. Lubana et al. (2025) proposed a framework for studying emergent capabilities using context-sensitive grammars and compositional tasks, under the theory of bipartite graph percolation. While our scope and domain are different, we show that the framework of graph percolation still applies and can explain the learning dynamics observed in our setup, hinting that learning token-character correspondence is a similar problem to learning correspondences between concepts and their properties.

## 3 Method

### 3.1 Tokenization-Induced Information Bottleneck

Let $\Sigma$ be a character alphabet and let $C = (c_1, \ldots, c_n) \in \Sigma^n$ denote the character sequence (spelling) of a word. Let $W$ denote the lexical identity (word type) determined by $C$: there is a deterministic map $g : \Sigma^* \to \mathcal{W}$ with $W = g(C)$. Let $X$ be the corpus context, all surrounding tokens excluding the word at the current position. The quantity of interest is the empirical mutual information (Shannon and Weaver, 1998):

$$\hat{I}(X; C) = \mathbb{E}\left[\log \frac{p(x, c)}{p(x)\, p(c)}\right] \tag{1}$$

Namely, how much the context $X$ tells us about the characters $C$ of the word that appears in that context. Because $W$ is a deterministic function of $C$, we have $I(W; C) = H(C) = H(W)$. Moreover, natural text exhibits a certain phenomenon: humans do not need to explicitly mention the characters in a word, so the context $X$ provides little signal about $C$, which means the empirical

mutual information $\hat{I}(X;C)$ is low, even though the theoretical mutual information $I(W;C)$ is high, since $W$ deterministically determines $C$ (i.e., $\hat{I}(X;C) \ll I(W;C)$). Thus, for a language model trained only on tokenized word sequences, the empirical mutual information $\hat{I}(X;C) \approx 0$, unless the model is explicitly character-aware. Similar to the presence of commonsense facts, this data sparsity reflects a form of reporting bias (Shwartz and Choi, 2020): humans do not encode character-level details in natural text, leading to under-representation of this information in the training signal. However, learning commonsense knowledge requires explicit data collection (Speer et al., 2017), whereas learning character-token correspondences is comparatively simpler: we know at all times which characters comprise a word, and we can leverage this in the design of a character-aware architecture.

### 3.2 Experimental Setup

**Word-level and Character-level tasks.** Previous works explored pretrained LLMs' performance on several character-level tasks (Shin and Kaneko, 2024; Zhang and He, 2024) and show that their performance is sub-par. In this work, we create a set of 7 word-level tasks and 12 character-level tasks to systematically explore emergent capabilities for character manipulation across training. Compared to previous works, our tasks are easier and do not involve counting or multi-hop reasoning (e.g., count vowels of every even word). Table 1 shows our tasks alongside examples for parameters, inputs, and desired outputs. By design, the tasks have input-output combinations tokenized either as words $\leftrightarrow$ words (all word-level tasks), characters $\leftrightarrow$ words (dirty-input character tasks), words $\leftrightarrow$ characters (clean-input character tasks), or a mix of tokenizations (e.g., "Rewrite uppercase" / "Replace letters"). As such, for character-level tasks, multi-character tokens might be imperfectly split into characters (e.g., "Remove letter"), and models are forced to indirectly learn token-character correspondence across many training steps. Tasks are evaluated using an exact match between the model output and the desired output. While using exact match metrics impacts evaluation curves (Schaeffer et al., 2023), they are correlated with other softer metrics such as log-probabilities, and inflection points between memorization and generalization phases match between the two (Lubana et al., 2025). Furthermore, an exact match enables us to compare performance unambiguously across different tokenizers and vocabulary sizes.

**Vocabulary construction.** We opted for a strictly controlled and synthetic experimental environment to test the capability of tokenized language models to learn character-level tasks and to eliminate as many confounding factors as possible. We generate a fixed-length vocabulary of words $V$, which is comprised of all single-character letters, including uppercase, numbers, and a space character. Multi-character tokens are all comprised of the same number of characters, $K$, uniformly sampled (see Appendix A: Table 3). In our work, $K \in \{4, 6, 8\}$ and $|V| \in \{2^8, \ldots, 2^{15}\}$. To encode a task, we use a special task token for each task, which is optionally followed by parameters (see Table 1). Consequently, our tokenizer is comprised of single characters, numbers, and multi-character words, each with its unique ID. As such, if a multi-character word is corrupted, it will be represented through individual characters as a fallback tokenization, with no intermediate subwords.

For our analysis, we ignore language grammar, since none of the tasks require grammar manipulation, only token and character manipulation. To construct sentences, words are sampled uniformly from the tokenizer vocabulary (see Appendix A: Table 4), ignoring the Zipfian distribution of real-world languages (Piantadosi, 2014). However, we also test performance on randomly sampled sentences from Wikipedia, using two pretrained tokenizers (i.e., GPT-2 (Radford et al., 2019) and LLaMA-2 (Touvron et al., 2023)). It is expected that real-world texts would not qualitatively change learning dynamics; still, they would make learning harder due to the imbalanced distribution of characters per word and different word lengths.

In this most simplified version of the problem, the only factor influencing model performance is its ability to connect tokens with their characters, which appear fragmented and inconsistent across training. Such a strictly controlled environment is similar to other concurrent works (Allen-Zhu and Li, 2023) aiming to explain model capabilities without real-world confounders. In this setup, the model is forced to learn the algorithm behind each

| | Task Name | Example Param. | Example Input | Example Output |
|---|---|---|---|---|
| Word-Level | Remove word | red | Strawberries are red and sweet. | Strawberries are and sweet. |
| | Remove word every K | 2 | Strawberries are red and sweet. | Strawberries red sweet. |
| | Swap every K words (clean) | 5 | Strawberries are red and sweet. | sweet. are red and Strawberries |
| | Swap every K words (dirty) | 5 | sweet. are red and Strawberries | Strawberries are red and sweet. |
| | Replace words | are, red | Strawberries are red and sweet. | Strawberries red red and sweet. |
| | Reverse the words (clean) | N/A | Strawberries are red and sweet. | sweet. and red are Strawberries |
| | Reverse the words (dirty) | N/A | sweet. and red are Strawberries | Strawberries are red and sweet. |
| Character-Level | Remove letter | r | Strawberries are red and sweet. | Stawbeies ae ed and sweet. |
| | Rewrite uppercase every K letters | 3 | Strawberries are red and sweet. | StrAwbErrIes arE rEd And swEet. |
| | Replace letters | e, s | Strawberries are red and sweet. | Strawbsrriss ars rsd and swsst. |
| | Rewrite with every K letter | 3 | Strawberries are red and sweet. | Saei eea e. |
| | Swap every K letters (clean) | 2 | Strawberries are red and sweet. | tSarbwreirsea err dea dns ewte. |
| | Swap every K letters (dirty) | 2 | tSarbwreirsea err dea dns ewte. | Strawberries are red and sweet. |
| | Remove letter every K | 4 | Strawberries are red and sweet. | Strwberie ar re an swet. |
| | Rewrite uppercase every K words | 2 | Strawberries are red and sweet. | STRAWBERRIES are RED and SWEET. |
| | Reverse words (clean) | N/A | Strawberries are red and sweet. | seirrebwartS era der dna .teews |
| | Reverse words (dirty) | N/A | seirrebwartS era der dna .teews | Strawberries are red and sweet. |
| | Reverse (clean) | N/A | Strawberries are red and sweet. | .teews dna der era seirrebwartS |
| | Reverse (dirty) | N/A | .teews dna der era seirrebwartS | Strawberries are red and sweet. |

Table 1: Summary of proposed tasks used in our work. The tasks address either word-level or character-level manipulations and optionally require input parameters.

task, through so-called "induction heads" (Olsson et al., 2022) or "name mover heads" (Wang et al., 2022), since the tasks only require token-level manipulations and not semantic understanding (Shin and Kaneko, 2024).

### 3.3 Generating tokens by attending to characters.

We design a lightweight character-aware module that complements the main Transformer decoder to increase the mutual information between tokens and their constitutive characters. In our design, we were guided by several criteria: *(i)* the character-aware module must be lightweight *(ii)* the model output type must remain unchanged (i.e., still output multi-character tokens), *(iii)* there is an unambiguous correspondence between tokens and their characters, and *(iv)* there is an unambiguous order of characters inside a token.

Figure 2 showcases our architecture. Given these criteria, we designed a small, 1-layer Transformer block that uses a *Block-Causal Self-Attention* mask to process characters. Since the main model operates on multi-character tokens, whenever a new token is generated, the character model has access to all its characters, removing the need for a casual diagonal attention matrix. The block-causal attention mask enables the module to attend to all characters in the current token, as well as previous characters from previous tokens, but does not "cheat" by attending to the characters of future tokens. The order of characters inside a token is encoded using learnable *Intra-Token Position* embeddings, similar to Abacus Embeddings (McLeish et al., 2024). The dimensionality of the character encoder can be made smaller than the main module (in our case, $d_{chars} = \frac{1}{2} d_{tokens} = 256$). We also experiment with smaller dimensionalities for the character module (i.e., $d_{chars} \in \{64, 128, 256\}$, corresponding to ratios $\frac{d_{chars}}{d_{tokens}} \in \{\frac{1}{4}, \frac{1}{8}, \frac{1}{16}\}$). After encoding characters, the resulting embeddings interact with the token embeddings through a *Block-Causal Cross-Attention* operation at each layer of the main model. We also experiment with adding the character embeddings at a single layer of the main model, at different positions. The cross-attention operation prevents tokens from attending to future characters and ensures that each token attends to its corresponding characters alongside characters from previous tokens. Character-token correspondence is ensured through learnable *Inter-Token Position* embeddings.

**Overview.** The model is efficient by operating on multi-character tokens and not directly predicting characters, leveraging the tokenizer compression, and has explicit knowledge of the character composition of each token. In principle, this design can be extended hierarchically, for example, hav-

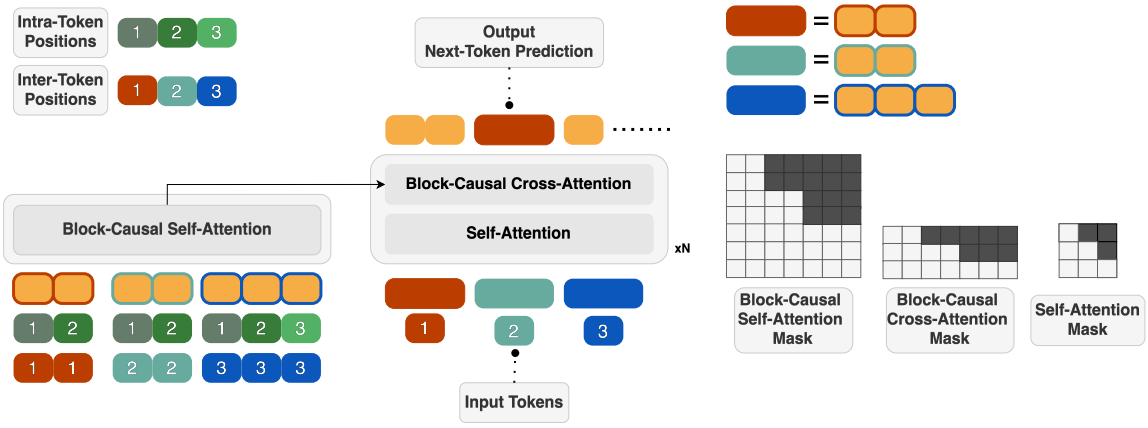


Figure 2: Diagram of our character-aware language model. During inference, each token attends to its corresponding characters using a block-causal cross-attention operation. Characters are encoded alongside their positions within their corresponding tokens using a small 1-block Transformer decoder, using a block-causal self-attention mechanism. MLPs are omitted in the figure for brevity.

ing tokens attend to their constituent subwords and each subword attending to its constituent characters. While the character module is significantly smaller than the main module, it still suffers from the quadratic complexity of the attention operation. Presumably, the character encoder can be made more efficient to avoid quadratic attention by utilizing, for example, local attention patterns (Beltagy et al., 2020) or by using more specialized modules such as linear recurrent units (Orvieto et al., 2023). Our model is reminiscent of other works in computer vision, such as CrossViT (Chen et al., 2021), and is part of a larger pattern of designing architectures that use hierarchical representations (Nawrot et al., 2021; Chalkidis et al., 2022; He et al., 2024). Nevertheless, this pattern is more common in computer vision than in NLP. This hierarchical character-to-token cross-attention design addresses the problem of "perception" of current LLMs, which capture high-level semantic meaning, but struggle at "high-resolution", in terms of perceiving individual characters of each token.

### 3.4 Training configuration

All models were trained on a uniform sample over all tasks for 750k iterations. We used Adam (Kingma and Ba, 2014) optimizer, a batch size of 64, and a learning rate of 0.00001, annealed using a cosine decay scheduler (Loshchilov and Hutter, 2016). The baseline model has 10M parameters, excluding embedding matrices, across 8 layers, with a model dimensionality of 512. Similarly, our model has 11M parameters, with 1M being allocated to the character encoder. The character encoder is a lightweight, single-block Transformer with a dimensionality of 256. One important advantage of our experimental setup is that it is readily reproducible on a single A100 GPU. Training took approximately one day per run for all ∼60 runs. Models were trained with "infinite" data, since input sentences and tasks were generated on-the-fly. In the case of training on sentences from Wikipedia, we pre-generated 5M sentences. In our experiments, every hyperparameter is kept fixed, except for the vocabulary and the tokenizer.

## 4 Experiments & Results

In Figure 4, we show the evolution of average accuracy across character tasks for both the base language model and our model that incorporates character information. The emergence point for character understanding tasks is progressively offset as a function of vocabulary size $|V|$ and number of characters per token $K$. In contrast, for our model, the emergence points are stable across $|V|$ and $K$, having reasonable performance gain even in scenarios where the base model has accuracy equal to 0. This effect is also present, although not as prominently, for token understanding tasks (Figure 5), since token manipulation tasks can be easily learned by the base model.

In Figure 3 we show the emergence step across vocabulary sizes. We plot the training step at which

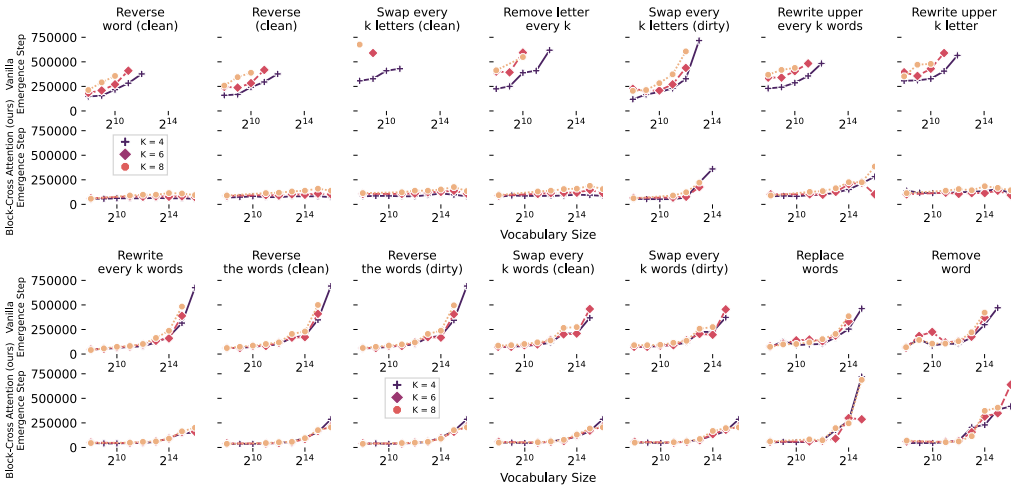


Figure 3: Emergence point for acquisition of character-level *(top)* and word-level *(bottom)* understanding across vocabulary sizes for the "vanilla" model and our character-infused model. For our model, capabilities emerge early on in training for character-level tasks, and do not depend on $|V|$ or $|K|$.

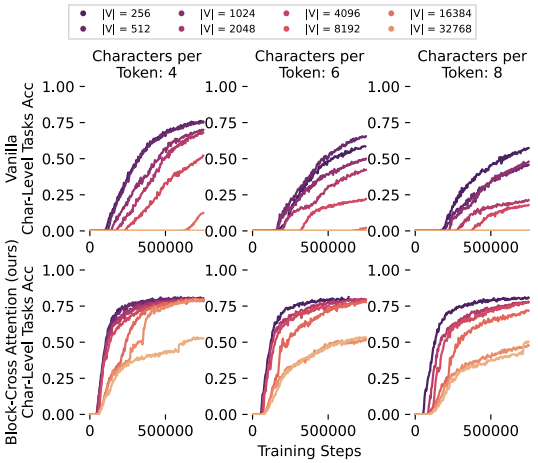


Figure 4: Evolution of average accuracy over character-level tasks. Using standard transformer decoder *(top)*, the emergence point depends on the vocabulary size and $K$, whereas our architecture *(bottom)* eliminates the differences in emergence points across vocabulary sizes.

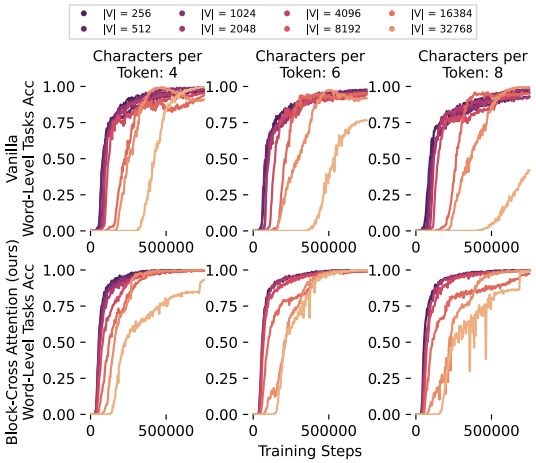


Figure 5: Evolution of average accuracy over word-level tasks. Emergence points for word-level tasks are not affected by vocabulary size as prominently as character-level tasks. While our architecture *(bottom)* targets character-token associations, it still provides improvement by increasing the amount of information per training sample compared to the standard LM *(top)*.

the accuracy for a task is larger than 0.5%, across training runs of different $|V|$ and $K$. We find that increasing vocabulary size is correlated with a later emergence point for the vanilla model for both word-level and character-level tasks, having a predictable relationship. The addition of the character-aware module eliminates the dependence on $|V|$ for character-level tasks, but to a lesser extent for the word-level tasks.

### 4.1 Evaluation on real-world data

We trained the baseline language model and our model on sentences sourced from Wikipedia, using two pretrained tokenizers (i.e., GPT-2 (Radford et al., 2019) and LLaMa-2 (Touvron et al., 2023)), with vocabulary sizes of 50K and 32K tokens, respectively. In Figure 6 we show our results: incorporating characters has a significant effect on learning dynamics for both tokenizers, with the

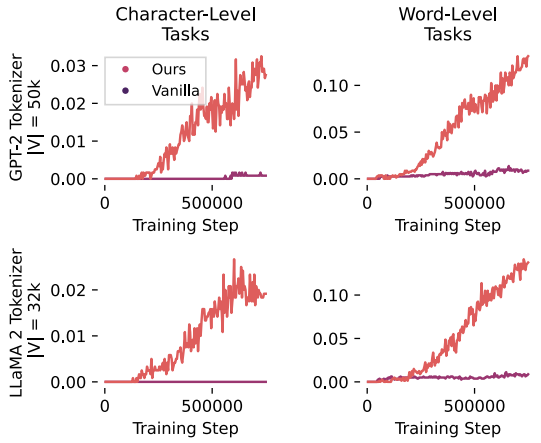


Figure 6: The effect of using real sentences sourced from Wikipedia in evaluating character understanding tasks, across two pretrained tokenizers.

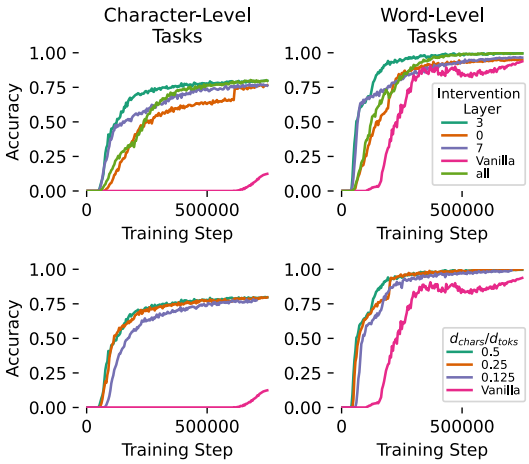


Figure 7: The effect of the position of the block-cross attention in the main model *(top)* and that of downsizing the char. encoder *(bottom)*. $|V| = 8192$ and $K = 4$.

base model being unable to learn character composition of words across training. As our results point to, this effect will be amplified with larger vocabularies: current LLMs tend to benefit from having progressively larger vocabularies (Huang et al., 2025), with models such as Gemma 3 (Team et al., 2025) operating on a vocabulary of 256K tokens, which also implies more characters per token. Our results indicate that character understanding tasks in tokenized language models are a form of "inverse scaling" (McKenzie et al., 2023): the larger the tokenizer vocabulary, the slower the model learns.

### 4.2 The effect of downsizing the character encoder

In Figure 7, we show results for varying the position of the cross-attention operation in the main model by incorporating character information either at the beginning, the middle, or the end of the language model. Similarly, we reduce the dimensionality of the character encoder to 12.5% of that of the main model, making the character model fast and lightweight in terms of memory consumption. Infusing character information in the middle of the model yields the best results, and downsizing the character encoder does not significantly alter the training curves, suggesting that only the presence of characters and their association with tokens is sufficient.

### 4.3 The effect of increasing base model size

We scaled the model in a principled way using maximal update parametrization ($\mu$P (Yang et al., 2022)) to increase the model width (size of the linear layer) through a multiplicative factor. Using $\mu$P ensures that the hyperparameters for the smaller model (e.g. learning rate) can be directly transferred to larger versions of the same model, making the comparison fair between scales in terms of optimal hyperparameters. We scaled the base encoder, but kept the character encoder fixed. The results presented in Table 2 show that scaling the width of the model reduces the emergence point, but without addition of the character encoder, the acquisition is slow and inefficient. In other words, without the character encoder, model needs $\times 15$ more parameters to reach the same performance. Across scales, the size of the character encoder is negligible. These results are for our simplified case, assuming uniform distribution of characters and tokens. Our other results (Figure 6) shows that this effect is greatly amplified with real world data.

### 4.4 A percolation model of character understanding.

In the interest of explaining the offset emergence points for the acquisition character understanding, we applied a percolation model of capability emergence, as described by Lubana et al. (2025). Readers are referred to the original work for a detailed explanation of this framework, which the authors

| Vocab. Size | Model Width | Base Params | Character-Aware | Params Char. Encoder (% of base) | Char. Tasks Emergence Step↓ |
|---|---|---|---|---|---|
| 8192 | 512 | 10M | ✗ | – | >750k |
| | | | ✔ | 1M (9%) | **77k** |
| | 1024 | 45M | ✗ | – | 180k |
| | | | ✔ | 1M (2%) | **53k** |
| | 2048 | 150M | ✗ | – | 81k |
| | | | ✔ | 1M (0.6%) | **20k** |
| 16384 | 512 | 10M | ✗ | – | >750k |
| | | | ✔ | 1M (9%) | **110k** |
| | 1024 | 45M | ✗ | – | 260k |
| | | | ✔ | 1M (2%) | **53k** |
| | 2048 | 150M | ✗ | – | 110k |
| | | | ✔ | 1M (0.6%) | **45k** |

Table 2: The effect of increasing model scale in terms of width. The addition of a small character-aware module substantially reduces the emergence point for the acquisition of character-level understanding. We kept $K = 4$ fixed across all runs.

applied in the context of learning concept-property relationships. In that scenario, emergence points coincided with a critical threshold $p_c$ in which the bipartite graph of concepts and their respective properties ($K$) is fully connected: across training, the model progressively learns edges between concepts and properties until reaching a certain threshold, proportional to $\sqrt{|K|}$, after which the model enters a sudden generalization phase. In our scenario, we have a direct analogy to concept learning: our "concepts" are multi-character tokens, and their "properties" are the set of characters they are composed of. As such, the emergence point should be proportional to the number of edges (Newman et al., 2001; Cohen et al., 2002; Lubana et al., 2025), in our case equaling $\sqrt{|V| * k}$. In Figure 8 we show the emergence points for the base language model. Scaling the training steps by $\sqrt{|V| * k}$ results in the collapse of the emergence points. This result indicates no conceptual difference between learning concept-property mappings and learning token-character mappings.

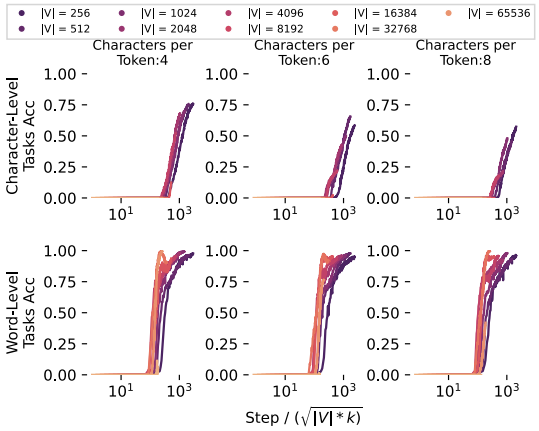


Figure 8: Graph percolation explains emergence points for acquisition of character understanding, similar to concept percolation (Lubana et al., 2025). Scaling training curves by the percolation threshold $\sqrt{|V| * k}$ collapses emergence points across vocabulary sizes.

## 5 Conclusions

Tokenization is crucial in language modeling, enabling long context and aiding generalization (Rajaraman et al., 2024). In this paper, we show that for a class of problems that require fine-grained understanding of character composition of tokens, models acquire such information very slowly, predictably dependent on the vocabulary size and number of characters per token. We argued that this is due to non-reporting bias and that this phenomenon is similar to learning commonsense facts from general text. There is a design mismatch in the way in which humans hierarchically perceive written text (from lines, characters, words and phrases) and the way LLMs process text.

To this end, we proposed a lightweight and straightforward architectural modification that eliminates this dependence on vocabulary size and showed that capabilities emerge faster and consistently. Lastly, we applied a theory of capability emergence in concept learning (Lubana et al., 2025) and showed that it applied to our setting, equating the phenomena of learning concepts with learning characters' composition in tokens.

## Limitations

The main limitation of our work is that we conducted most of our experiments in a synthetic and idealized setup to understand the phenomena of character understanding of tokens, without confounding factors. Nonetheless, our proposed architecture showed good results when training on real data, but its impact to real-world scenarios needs to be validated at larger scales.

## Acknowledgments

This research was supported by the project "Romanian Hub for Artificial Intelligence - HRIA", Smart Growth, Digitization and Financial Instruments Program, MySMIS no. 334906. We also thank Oleg Szehr for his unforgiving feedback that led to a more rigorous presentation of our method.

## References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Zeyuan Allen-Zhu and Yuanzhi Li. 2023. Physics of language models: Part 1, learning hierarchical language structures. *arXiv preprint arXiv:2305.13673*.

P. W. Anderson. 1972. More is different. *Science*, 177(4047):393–396.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv:2004.05150*.

Yekun Chai, Yewei Fang, Qiwei Peng, and Xuhong Li. 2024. Tokenization falling short: On subword robustness in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1582–1599, Miami, Florida, USA. Association for Computational Linguistics.

Ilias Chalkidis, Xiang Dai, Manos Fergadiotis, Prodromos Malakasiotis, and Desmond Elliott. 2022. An exploration of hierarchical attention transformers for efficient long document classification. *arXiv preprint arXiv:2210.05529*.

Chun-Fu Richard Chen, Quanfu Fan, and Rameswar Panda. 2021. Crossvit: Cross-attention multi-scale vision transformer for image classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 357–366.

Reuven Cohen, Daniel Ben-Avraham, and Shlomo Havlin. 2002. Percolation critical exponents in scale-free networks. *Physical Review E*, 66(3):036113.

Quyet V Do, Junze Li, Tung-Duong Vuong, Zhaowei Wang, Yangqiu Song, and Xiaojuan Ma. 2024. What really is commonsense knowledge? *arXiv preprint arXiv:2411.03964*.

Jonathan Gordon and Benjamin Van Durme. 2013. Reporting bias and knowledge acquisition. In *Proceedings of the 2013 Workshop on Automated Knowledge Base Construction*, AKBC '13, page 2530, New York, NY, USA. Association for Computing Machinery.

Haoyu He, Markus Flicke, Jan Buchmann, Iryna Gurevych, and Andreas Geiger. 2024. HDT: Hierarchical document transformer. In *First Conference on Language Modeling*.

Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and Xun Zhou. 2025. Over-tokenized transformer: Vocabulary is generally worth scaling. *Preprint*, arXiv:2501.16975.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. 2020. Compositionality decomposed: How do neural networks generalise? volume 67, pages 757–795.

Md Mofijul Islam, Gustavo Aguilar, Pragaash Ponnusamy, Clint Solomon Mathialagan, Chengyuan Ma, and Chenlei Guo. 2022. A vocabulary-free multilingual neural tokenizer for end-to-end task learning. *arXiv preprint arXiv:2204.10815*.

Guy Kaplan, Matanel Oren, Yuval Reif, and Roy Schwartz. 2025. From tokens to words: On the inner lexicon of LLMs. In *The Thirteenth International Conference on Learning Representations*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Ilya Loshchilov and Frank Hutter. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*.

Ekdeep Singh Lubana, Kyogo Kawaguchi, Robert P. Dick, and Hidenori Tanaka. 2025. A percolation model of emergence: Analyzing transformers trained on a formal language. In *The Thirteenth International Conference on Learning Representations*.

Ian R. McKenzie, Alexander Lyzhov, Michael Martin Pieler, Alicia Parrish, Aaron Mueller, Ameya Prabhu, Euan McLean, Xudong Shen, and 1 others. 2023. Inverse scaling: When bigger isn't better. *Transactions on Machine Learning Research*. Featured Certification.

Sean Michael McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and 1 others. 2024. Transformers can do arithmetic with the right embeddings. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. 2021. Hierarchical transformers are more efficient language models. *arXiv preprint arXiv:2110.13711*.

Pit Neitemeier, Björn Deiseroth, Constantin Eichenberg, and Lukas Balles. 2025. Hierarchical autoregressive transformers for tokenizer-free language modelling. In *The Thirteenth International Conference on Learning Representations*.

Allen Newell. 1983. *Intellectual issues in the history of artificial intelligence*, page 187294. John Wiley & Sons, Inc., USA.

Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. 2001. Random graphs with arbitrary degree distributions and their applications. *Physical review E*, 64(2):026118.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, and 1 others. 2022. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. 2023. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pages 26670–26698. PMLR.

Core Francisco Park, Maya Okawa, Andrew Lee, Ekdeep Singh Lubana, and Hidenori Tanaka. 2024. Emergence of hidden capabilities: Exploring learning dynamics in concept space. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Steven T Piantadosi. 2014. Zipf's word frequency law in natural language: a critical review and future directions. *Psychon Bull Rev*, 21(5):1112–1130.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Nived Rajaraman, Jiantao Jiao, and Kannan Ramchandran. 2024. Toward a theory of tokenization in llms. *arXiv preprint arXiv:2404.08335*.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. 2023. Are emergent abilities of large language models a mirage? In *Thirty-seventh Conference on Neural Information Processing Systems*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Claude E Shannon and Warren Weaver. 1998. *The mathematical theory of communication*. University of Illinois press.

Andrew Shin and Kunitake Kaneko. 2024. Large language models lack understanding of character composition of words. In *ICML 2024 Workshop on LLMs and Cognition*.

Vered Shwartz and Yejin Choi. 2020. Do neural language models overcome reporting bias? In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6863–6870.

Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31.

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, and 1 others. 2022. Charformer: Fast character transformers via gradient-based subword tokenization. In *International Conference on Learning Representations*.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. 2024. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M Rush. 2024. Mambabyte: Token-free selective state space model. In *First Conference on Language Modeling*.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2022. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, Yitao Liang, and Team CraftJarvis. 2023. Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Chuhan Wu, Fangzhao Wu, Tao Qi, and Yongfeng Huang. 2021. Hi-transformer: Hierarchical interactive transformer for efficient and effective long document modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 848–853.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. 2022. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *Preprint*, arXiv:2203.03466.

Yidan Zhang and Zhenan He. 2024. Large language models can not perform well in understanding and manipulating natural language at both character and word levels? In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 11826–11842.

# A Appendix

```python
def make_vocab(vocab_size, K):
    vocab = dict()
    token_id = 0

    numbers = '0123456789'
    letters_lower = 'abcdefghijklmnopqrstuvwxyz'
    letters_upper = letters_lower.upper()
    letters = letters_lower + letters_upper

    # adding the bytes and space
    for token_name in letters + numbers + ' ':
        vocab[token_name] = token_id
        token_id += 1

    # adding multi-character tokens
    for _ in range(vocab_size):
        word = ''.join(random.choice(letters) for _ in range(K))
        vocab[word] = token_id
        token_id += 1

    return vocab
```

Table 3: Python snippet for tokenizer vocabulary creation. In our work, words are made from randomly sampled characters of fixed size $K$. The tokenizer vocabulary contains single-characters, numbers, multi-character words, and a space as individual unique tokens. Additionally, each task is represented as a unique special token.

```python
def make_task(tokenizer):
    target_task = random.choice(TASKS)
    # choose multi-character words
    word_list = [
        w for w in tokenizer.vocab2id.keys()
        (if len(w) > 1 and w not in SpecialTokens)
    ]
    random_sentence = ' '.join([
        random.choice(word_list) for _ in range(16)
    ])
    task_output = target_task(random_sentence)
    # task_token, param, input, output
    return task_output
```

Table 4: Python snippet for task creation. In our work, a sentence is comprised of 16 uniformly sampled multi-character words from the tokenizer, upon which the task algorithm is enacted.