LLM AUGMENTED LLMs:
EXPANDING CAPABILITIES THROUGH COMPOSITION

Rachit Bansal\(^1\) Bidisha Samanta\(^1\) Siddharth Dalmia\(^2\) Nitish Gupta\(^1\) Shikhar Vashishth\(^1\)
Sriram Ganapathy\(^1\) Abhishek Bapna\(^2\) Prateek Jain\(^1\) Partha Talukdar\(^1\)
\(^1\)Google Research India \(^2\)Google DeepMind

ABSTRACT

Foundational models with billions of parameters which have been trained on large corpora of data have demonstrated non-trivial skills in a variety of domains. However, due to their monolithic structure, it is challenging and expensive to augment them or impart new skills. On the other hand, due to their adaptation abilities, several new instances of these models are being trained towards new domains and tasks. In this work, we study the problem of efficient and practical composition of existing foundation models with more specific models to enable newer capabilities. To this end, we propose CALM—Composition to Augment Language Models—which introduces cross-attention between models to compose their representations and enable new capabilities. Salient features of CALM are: (i) Scales up LLMs on new tasks by ‘re-using’ existing LLMs along with a few additional parameters and data, (ii) Existing model weights are kept intact, and hence preserves existing capabilities, and (iii) Applies to diverse domains and settings. We illustrate that augmenting PaLM2-S with a smaller model trained on low-resource languages results in an absolute improvement of up to 13\% on tasks like translation into English and arithmetic reasoning for low-resource languages. Similarly, when PaLM2-S is augmented with a code-specific model, we see a relative improvement of 40\% over the base model for code generation and explanation tasks—on-par with fully fine-tuned counterparts.

1 INTRODUCTION

Large Language Models (LLMs) have shown to encompass a range of foundational capabilities such as commonsense and factual reasoning, world knowledge, and coherent language generation (Bubeck et al., 2023; Google et al., 2023). Leveraging these foundational capabilities, a number of efforts in the community have fine-tuned these models to enable domain-specific capabilities such as code generation, copy editing, and mathematical problem solving (Lewkowycz et al., 2022; Singhal et al., 2023). This has resulted in the development of several specialized large models with domain-specific capabilities. For example, there are models that do well on standard code generation but are not as proficient in general logical reasoning and vice-versa. Presence of such a large number of domain-specific models leads to a natural question: Can we compose an anchor model with a domain-specific augmenting model to enable new capabilities? For example, can we compose an augmenting model’s code understanding capability with an anchor LLM’s language generation capability to enable code-to-text generation capability?

The typical approach for this problem is to further pre-train or (efficiently) fine-tune the anchor model on the data that was originally used to train the augmenting model (Hu et al., 2021; Kessler et al., 2022). However, many a times such solutions are not feasible since training large models is computationally expensive, especially since the augmenting model itself may be an LLM trained on a massive corpora. Further, processing data from multiple sources might not be feasible due to privacy concerns and organizational boundaries. Working with multiple distinct models is also desirable since it allows the reuse of existing models with established capabilities, providing better control and avoiding catastrophic forgetting that is prevalent in conventional approaches.

\(^1\)Correspondence to Rachit and Bidisha: [brachit, bidishasamanta]@google.com
Since $x_1 = 10$, $x_8 = 14$, $x_n = 2$, $x_1 + x_8 * x_n = 38$

Figure 1: **Overview of CALM.** To augment an anchor LLM ($m_B$) with new capabilities through composition with a specialized augmenting model ($m_A$). Figure illustrates three $m_A$ with different capabilities: key-value mapping (left), low-resource languages (center), and code (right). Models $m_A$ and $m_B$ remain unchanged (⊗) during composition. A few additional parameters are learnt over models’ layer representations. Leftmost plot shows an $m_A$ trained on a set of string-integer mappings, e.g., $\{x_1 : 10, \ldots, x_n : 2\}$. $m_B$ is a large LM with arithmetic capabilities. CALM composes these two frozen models to solve the task of arithmetic on keys which either models could not solve on their own (§4.1). Notably, CALM generalizes to the entire key-value set despite training with arithmetic examples spanning only 20% of the keys.

To address the training and the data challenges mentioned above, we propose and study a practical setting for model composition: (i) we are given access to one (or more) augmenting model(s) and an anchor model, (ii) we are not allowed to modify the weights of either models, and (iii) we only have access to a small amount of data, representing the “combined skills” of the given models, e.g., code generation with complex logical reasoning.

Prior work has largely approached the question of composition from either a routing or a merging standpoint, neither of which provide an effective solution to capture this setting. Routing between the given models, i.e., choosing an output of one model over the other (Ma et al., 2019), or performing a soft ensemble (Muqeeth et al., 2023) is not effective when neither of the models can demonstrate the desired capability. Another body of work creates a combined model by an arithmetic combination of base model parameters (Wortsman et al., 2022; Ilharco et al., 2022; Matena & Raffel, 2022). However, these settings are naturally restrictive and their efficacy is unclear when combining models with different sizes and pre-training objectives (Yadav et al., 2023).

In this work, we propose a novel **Composition to Augment Language Models (CALM)** framework to address the general model composition setting mentioned above. Rather than a shallow combination of the augmenting and anchor LMs (Wortsman et al., 2022; Ilharco et al., 2022), CALM introduces a small number of trainable parameters over both augmenting and anchor models’ intermediate layer representations. CALM finds an effective combination of the given models to perform new challenging tasks more accurately than either of the models alone, while preserving the capabilities of individual models. Figure 1 highlights few motivating scenarios for CALM.

We study key practical applications of CALM: language inclusivity and code generation. For language inclusivity (§4.2), we use a model that has been trained on a set of low-resource languages. We observe that composing this model with the LLM allows us to borrow its generation and reasoning capabilities to achieve significantly better performance on translation and arithmetic reasoning tasks for low-resource languages (Tables 2 and 3). This composed model outperforms not only the two base models but also versions of the LLM that have been further pre-trained or LoRA (Hu et al., 2021) fine-tuned for the set of low-resource languages. For code generation (§4.3), we use a model that has been trained on open-source code across a variety of programming languages. Composing this model with the LLM—hence borrowing its low-level logic and generation capabilities—outperforms the two base models (Table 4) on code explanation and code completion tasks.
2 RELATED WORKS

Parameter efficient fine-tuning: A large body of prior work focuses on parameter efficient ways of fine-tuning models for new tasks by introducing a small number of trainable parameters, keeping the original model intact (Houlsby et al., 2019; Wang et al., 2020; Pfeiffer et al., 2021; Hu et al., 2021; Kessler et al., 2022). Since this paradigm allows a small set of new parameters to be trained, it is challenging to use these approaches to augment novel domains and knowledge sources that are entirely absent from the original training corpus. In contrast, CALM enables a model to be adapted to new domains using augmenting models. In Section 4.4, we draw empirical comparisons between CALM and LoRA (Hu et al., 2021), a representative parameter efficient fine-tuning method.

Model Merging: Merging different expert models with simple techniques like task vector averaging provides a way of recombining different capabilities of these models (Ilharco et al., 2022; Matena & Raffel, 2022). However, these methods are only relevant when the original models are well aligned. Other related approaches are also applicable only when the models are derived from the same model (Matena & Raffel, 2022) or they are of same size (Muqeeth et al., 2023). In contrast, CALM is more generic and is applicable to any set of models.

Model and Task Compositionality: The modular encoder-decoder based method in Dalmia et al. (2022) adapts components of encoder-decoder models to allow flexible re-usability of different encoders, each with their own capabilities. Several past studies explore compositionality of modality-specific encoders with language models to serve multi-modal use-cases (Ziegler et al., 2019; Alayrac et al., 2022). Typically, they introduce cross-attention parameters across a language model in order to attend to representations from an image encoder and show an effective transfer of modalities across models. In this work, we extend the ideology of model re-use and modularity to composition of capabilities in large language models.

Models as Tools: Another interesting direction for using multiple language models to solve a downstream task has been to perform composition in the models’ input text space (Zeng et al., 2022; Shen et al., 2023). Schick et al. (2023) have demonstrated how a model can be taught to use external tools—there might be an opportunity to investigate if other models can be called as a part of the same framework. Since these approaches require a large amount of prompt engineering, in this work we focus on composition through representations that can be learnt automatically.

3 COMPOSITION TO AUGMENT LANGUAGE MODELS (CALM)

Given an anchor model \( m_B \) and an augmenting model \( m_A \), CALM aims to compose the two models \( (m_A \oplus m_B) \) to enable new capabilities as a composition of capabilities of the two individual models.

As discussed in the introduction, we study this composition in a practical setting with the following assumptions: i) we can access weights, run forward and backward pass, and access intermediate representations of both \( m_B \) and \( m_A \), ii) we are not allowed to change weights of both the models, iii) we do not have access to the training data, hyperparameters, training states of both the base models, iv) we are provided a few examples from the target composition domain.

The goal is to learn a composition \( m_A \oplus m_B = f(m_A, m_B, \Theta_C, D_C) \) to achieve some joint task C. The weights of \( m_A \) and \( m_B \) are frozen. \( \Theta_C \) is the additional set of trainable parameters introduced to learn the composition and \( D_C \) refers to the set of examples that are used to learn this composition.

3.1 LEARNING TO COMPOSE (\( \Theta_C \))

As outlined in Figure 1, we operate over a selected set of layers from \( m_B \) and \( m_A \) at all times. We learn two sets of additional parameters over these layers: (i) A simple set of linear transformations, \( f_{proj}(\cdot) \) that maps an \( i^{th} \) layer representation from \( m_A \) to the dimensionality of representations from \( m_B \), and (ii) A set of cross-attention layers, \( f_{cross}(\cdot, \cdot) \) that cross-attend between this transformed layer representation and a \( j^{th} \) layer representation from \( m_B \).

Compositional Layers: Let the augmenting model \( m_A \) and the anchor model \( m_B \) have \( N_A \) and \( N_B \) layers, respectively. Also, let \( D_A \) and \( D_B \) be the token dimensionality of the two models. We
first choose a set of compositional layers—$\mathbb{L}_A$ and $\mathbb{L}_B$—for both models, over which the set of new learnable parameters are introduced during composition. If $n_A = |\mathbb{L}_A|$ and $n_B = |\mathbb{L}_B|$. For simplicity, we set $n_A = n_B = n$ and the gap between two contiguous selected layers is kept uniform based on the number of selected layers—that is, $(l_2 - l_1) = \cdots = (l_n - l_{n-1}) = N/n$. Further, $H_m \in \{H_{A1}, H_{A2}, \ldots, H_{AnA}\}$ denote the layer representation of a given input after each layer in $L_A$.

**Learned Projections:** Next we map representations from $m_A$ to that of $m_B$ via a projection layer. In particular, for each layer in $\mathbb{L}_A$, we learn a projection function $f_{\text{proj}} : \mathbb{R}^{D_m} \rightarrow \mathbb{R}^{D_B}$, that projects representations from these layers to the desired representation size of $m_B$. Let,

$$f_{\text{proj}}(H_m) \leftarrow \{f_{\text{proj}}(H_{A1}), f_{\text{proj}}(H_{A2}), \ldots, f_{\text{proj}}(H_{AnA})\}$$

This transformation enables cross-attention across models, and also performs an alignment of representations from $m_A$ and $m_B$ despite frozen weights of the base models.

**Cross-attention Layers:** Similar to the multi-headed cross-attention in encoder-decoder models (for example Vaswani et al. (2017) and Raffel et al. (2020))—we introduce cross-attention between representations of the anchor and the augmenting model. In particular, we use $f_{\text{proj}}(H_{Ai})$ from the augmenting model as the key and value vectors for each head in cross-attention. We use the vector $H_{Bj}$ from the anchor model as the query vector, which leads to the following cross-attention setup:

$$f_{\text{cross}}(f_{\text{proj}}(H_{Ai}), H_{Bj}) = \text{Concat}_k \ (\text{head}_k \ W^Q) \ \forall k \in \mathbb{H}_H$$

where, $\text{head}_k = \text{Attn}_k(Q_B, K_A, V_A)$,

and, $Q_B = H_{Bj} W^Q, K_A, V_A = f_{\text{proj}}(H_{A1}) W^K, f_{\text{proj}}(H_{Ai}) W^V$

Here, $\mathbb{H}_H$ represents the number of attention heads used for cross-attention which, in our case, is typically the same as the number of heads used for self-attention in $m_B$. Each of $W^Q \in \mathbb{R}^{D_B \times D_B}$, $W^K \in \mathbb{R}^{D_B \times D_A}$, and $W^V \in \mathbb{R}^{D_B \times D_B/\mathbb{H}_H}$ are learnable weight matrices, where $k \in \{1..\mathbb{H}_H\}$.

Finally, the cross-attention output is added as a residual connection to the layer representations of $m_B$. The resultant output vector, in-turn, is the input to the succeeding layer in $m_B$:

$$H_{A \oplus Bj} = H_{Bj} + f_{\text{cross}}(f_{\text{proj}}(H_{Ai}), H_{Bj})$$

Here, $H_{A \oplus Bj}$ denotes the input to the $(j + 1)^{th}$ layer of the composed model. All layers in $L_A$ and $L_B$ are utilized in a similar manner. Propagating over the remaining layers in $m_B$ gives us a final output token $y_t$, decoded for the $t^{th}$ timestep. Akin to usual auto-regressive decoding, the output token for each time-step is appended to the input: $x_{t+1} = x_t \oplus y_t$. Since the updated input at each time step is passed to both models, all representations for the two models are refreshed.

### 3.2 Composition Training Data ($D_C$)

Since the target model $m_{A\oplus B}$ involves a composition over the two models $m_A$ and $m_B$, we construct the set of training examples $D_C$ to depict a “combined skill” that enables $\Theta_C$ to attend over the two models appropriately for the target task.

Ideally, if the set of tasks involved in composition task are distinguished as $t_1$ and $t_2$ respectively, then we design $D_C$ to depict a joint task $C$. For example, with respect to our synthetic key-value setup: our final task (C) is to perform arithmetic over a set of keys. The augmenting model $m_A$ is trained to learn the given key-value pairs (notated as task, $t_1$) and the anchor model $m_B$ is generic model that can perform numeric arithmetic well (task $t_2$). For learning the set of parameters $\Theta_C$ for composition, we consider $D_C$ to be arithmetic over a held-in set of keys (task C), encompassing combined skills from the two models. In contrast to fine-tuning approaches like LoRA (Hu et al., 2021) that would require the entire knowledge source (here, key-values) during training time, we find that training composition on only a fraction of the keys can generalize to the full set.

In other real world settings, a clear distinction in specializing tasks for each model might be difficult to formulate and hence defining a task that captures the combined skills can be challenging. We find
that using a set of examples that capture certain capabilities of the two models suffices, i.e., some rough notion of $t_{A∪B}$. For our language inclusivity task, we use a mixture of examples containing a small amount of low-resource language and high-resource language data.

**Composing multiple models:** Finally, we note that while the method has been presented for a setting with one anchor LLM and only one augmenting model, CALM is applicable to multiple augmenting models as well. In particular, CALM would require learning similar projection and cross-attention components between the anchor and each of the augmenting model. We leave a thorough investigation of this as a topic of future work.

## 4 Experiments

We demonstrate the following in three domains: (a) an anchor LLM ($m_A$) can be composed with an augmenting model ($m_B$) trained on mappings between string keys and number values to solve arithmetic expressions over those keys requiring both, knowledge of the KV mappings and arithmetic capabilities (§4.1); (b) how CALM can be used to expand the language coverage of an anchor LLM ($m_B$) to low-resource languages it has not seen during pre-training. We show that an augmenting model ($m_A$) pre-trained on low-resource languages can be composed with such an anchor model to significantly improve translation and math-word problem solving capabilities in low-resource languages (§4.2); (c) how code completion and explanation can be improved by composing an anchor LLM with an augmenting model ($m_A$) specializing in the code domain (§4.3).

In all experiments, we start with a PaLM2-XXS model and further train it on domain-specific data to arrive at an augmenting model ($m_A$) that is then kept frozen during composition. Note that no task specific training data was used to train CALM. We use PaLM2-XS or PaLM2-S models as the anchor LLM ($m_B$) that is also kept frozen during composition training. For all our experiments, we set $N_A/n = 4$, i.e., we perform composition using every 4th layer output from $m_A$. Correspondingly, layers from $m_B$ ($L_B$) are chosen such that $n_B = n_A = n$, hence $n_B = N_B/4$.

### 4.1 Key-Value Arithmetic

We first study the setting where we have a small augmenting LM that has been trained to memorize string-to-integer key-value (KV) mappings, and a large anchor LM that is capable of performing arithmetic over integers. We wish to use CALM to compose them and enable a new capability of solving arithmetic expressions containing those keys.

**Key-Value Domain Knowledge** We first generate a repository of KV pairs containing $N_{KV} = 25K$ pairs by sampling English strings of $2-6$ characters from the vocabulary of the PaLM2-XXS model and randomly assigning them unique integer values in the range $[1, N_{KV}]$. This constitutes the knowledge artifact, $D_{KV}$. We further generate a collection of arithmetic expressions ($D_{KV,EXP}$) containing addition (+), subtraction (−), and multiplication (×) operations between $3-6$ keys by randomly sampling keys from $D_{KV}$ and operations to perform between them. We generate three datasets:

(i) KV-Substitution ($D_{KV,SUBS}$): This dataset maps each expression in $D_{KV,EXP}$ to an expression where the keys are replaced by their corresponding values. For example, this dataset contains examples of the form ($<K_1>+<K_2>−<K_3>, 10 + 22 − 24$).

(ii) KV-Arithmetic ($D_{KV,MATH}$): This dataset maps each expression in $D_{KV,EXP}$ to the numeric value arrived at by solving the arithmetic expression when the keys would be replaced by the corresponding values. For example, examples in this dataset look like ($<K_1>+<K_2>−<K_3>, 8$).

(iii) Numeric-Arithmetic ($D_{NUM,MATH}$): This dataset maps the value substituted version of each expression in $D_{KV,EXP}$ to the numeric value arrived at by solving the arithmetic expression. For example, examples in this dataset look like ($10 + 22 − 24, 8$).

**Models** We obtain augmenting model $m_A$ by further training a pre-trained PaLM2-XXS model on $D_{KV,SUBS}$ to make it memorize the KV pairs in $D_{KV}$. Note that, training on $D_{KV,SUBS}$ does not teach this augmenting model how to solve arithmetic expressions. Next, we use a pre-trained PaLM2-XS model as the anchor model $m_B$. This model is capable of solving numeric expressions with decent performance (see Table 1). Note that, this model has no knowledge of the KV pairs in $D_{KV}$.
We now take examples from the KV-Substitution dataset $D_{KV-SUBS}$ that only span 20% of the keys in $D_{KV}$ to form the training data for composition ($D_{C}$). We use $D_{C}$ to compose the augmenting model ($m_{A}$) having knowledge of $D_{KV}$ and the pre-trained anchor model $m_{B}$ by training the composition parameters ($\Theta_{C}$) using CALM as explained in §3. Both $m_{A}$ and $m_{B}$ are kept unchanged.

**Evaluation Task** We evaluate the composed model $m_{A\oplus B}$ for its ability to solve arithmetic expressions containing keys from $D_{KV}$. Specifically, we evaluate on the subset of $D_{KV-MATH}$ dataset that does not contain expressions used in $D_{C}$ during training. This way, we are able to measure the composed model’s ability to generalize to keys beyond what was observed during training.

Table 1: Evaluation (accuracy (%)) for a synthetic key-value (KV) task. $m_{A}$ is trained to memorize the KV mappings while $m_{B}$ excels at arithmetic. We see that a composition $m_{A\oplus B}$ is able to perform arithmetic over held-out keys.

<table>
<thead>
<tr>
<th></th>
<th>$m_{A}$</th>
<th>$m_{B}$</th>
<th>CALM ($m_{A\oplus B}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{KV-SUBS}$</td>
<td>98.1</td>
<td>0.0</td>
<td>92.9</td>
</tr>
<tr>
<td>$D_{NUM-MATH}$</td>
<td>4.2</td>
<td>73.7</td>
<td>72.0</td>
</tr>
<tr>
<td>$D_{KV-MATH}$</td>
<td>0.7</td>
<td>0.0</td>
<td>84.3</td>
</tr>
</tbody>
</table>

As expected, the anchor model $m_{B}$ gets 0% accuracy on the KV-Substitution and KV-Arithmetic tasks as it has not seen any data from $D_{KV}$. However, it performs well (73.7%) on the Numeric-Arithmetic task demonstrating capability of arithmetic over numerals.

Lastly, we see that the composed model $m_{A\oplus B}$ is able to solve all tasks with high accuracy, especially the KV-Arithmetic task (84.3%) for which both the underlying models fail. This shows that the composed model is able to leverage the relevant capabilities from both the augmenting and anchor model to solve a complex task.

### 4.2 Low-resource Language Inclusivity

Table 2: Translation performance for XX to English direction on the FLORES-200 dataset (Costajussà et al., 2022): We show results for a subset of 10 low-resource languages. Note that the composed model $m_{A\oplus B}$ significantly outperforms both $m_{A}$ and $m_{B}$. On the complete language list, $m_{A\oplus B}$ outperforms both the underlying models for 175 of 192 languages (Appendix A; Figure 2). $m_{BNTL}$ represents a skyline where $m_{B}$ has been further pre-trained on $D_{NTL}$. The composed model achieves similar performance for a tiny fraction of the training cost.

In this section, we study if we can compose such a large anchor LM $m_{B}$ with a smaller augmenting LM $m_{A}$ that has been pre-trained on low-resource languages, to perform translation and math-word problem solving tasks presented in these low-resource languages.

**Low-resource Language Corpora** We use the long-tail language set and the associated corpora from the Next Thousand Languages (NTL) effort (Caswell et al., 2020; Bapna et al., 2022) as the
Table 3: Evaluations for grade-school mathematics (GSM) problems on low-resource (LRL) and high-resource (HRL) languages. We observe that CALM yields significant gains for both evaluation sets. Gains on the HRL set suggests that CALM avoids catastrophic forgetting.

domain data D_{NTL}. This large-scale corpora contains web-crawled monolingual sentences and translation pairs for ∼1000 languages. The dataset has been used for language expansion in translation systems and language models (Garcia et al., 2021; Siddhant et al., 2022).

Models Akin to §4.1, we obtain augmenting model $m_A$ by training the PaLM2-XXS model on $D_{NTL}$ to impart knowledge about these low-resource languages to the model. For $m_B$, we use the pre-trained PaLM2-S model. We use ∼5% of the same low-resource language corpora $D_{NTL}$ as the training data $D_C$ to compose $m_A$ and $m_B$ via CALM. Since both models are untrained during composition, the anchor model $m_B$ is not trained on any of the low-resource language data.

Evaluation Tasks We evaluate the composed model $m_{A\oplus B}$ on two tasks:
(i) Translating text from a non-English language to English: We carry out these evaluations in a 5-shot in-context learning paradigm on the FLORES-200 (Costa-jussà et al., 2022) dataset. This dataset contains examples for 200 high- and low-resource languages.
(ii) Performing grade school math word problems expressed in a non-English language: We evaluate on the multilingual version of the GSM-8K dataset (Shi et al., 2023) containing math word problems for English and 9 other high-resource languages. We further generated a silver-standard GSM-8K dataset for low-resource languages by automatically translating the English examples in GSM-8K to 25 low-resource languages supported by Google Translate.\(^1\)

Results Table 2 shows results on the FLORES-200 dataset (Costa-jussà et al., 2022), where the input is a low-resource (XX) language sentence and the output should be the corresponding English translation. For 10 low-resource languages shown in the Table, we see that both the underlying models $m_A$ and $m_B$ are outperformed by our composed model $m_{A\oplus B}$. We find that the composed model $m_{A\oplus B}$ outperforms $m_B$ on 175 of the complete set of 192 languages (Appendix A).

Table 3 shows the performance of these models on the grade-school math word problems from the GSM8K task (Cobbe et al., 2021) on low-resource languages (top) and high-resource languages (Shi et al. (2023); bottom). Firstly, we observe that the augmenting model $m_A$ does not perform well on this task due to its limited mathematical reasoning capabilities. On the other hand, the anchor model $m_B$ does much better given its mathematical reasoning capabilities and transfer-learning from high-resource languages. Finally, we observe that $m_{A\oplus B}$ outperforms both $m_A$ and $m_B$ on 18 of 25 low-resource and 9 of 10 high-resource languages, demonstrating effective composition of models.

\(^1\)We perform quality evaluations in Appendix 7.
Table 4: Evaluations for code generation and understanding across three tasks: Code Completion (CC), Text-to-Code (T2C), and Code-to-Text (C2T). Augmenting code understanding to \( m_B \) using \( m_A \) significantly improves performances across all datasets. \( m_{\text{Code}} \) represents a skyline where \( m_B \) further pretrained on the \( D_{\text{Code}} \), which shows catastrophic forgetting of text generation task.

See Table 6 (Appendix A.2) for a complete set of evaluations. Note that the last row in Table 3 shows that \( m_B \) when fine-tuned on \( D_{\text{NTL}} \) leads to worse performance than the pre-trained \( m_B \) indicating forgetting. Composing domain-specific model \( m_A \) with \( m_B \) using CALM avoids this.

### 4.3 Code Understanding and Generation

Code understanding and generation require two distinct types of capabilities: (a) knowledge of the syntax and semantics of code, and (b) knowledge of the world that the code is manipulating. While LLMs have a wealth of world knowledge, they could often lack the specific knowledge of code syntax due to a skewed representation of code data in their pretraining corpora. Conversely, small models trained specifically on code data could exhibit a good understanding of code syntax, but they may lack broad world knowledge and reasoning. CALM can enable best of both worlds.

#### Code Domain Data

Here, we use the code-specific corpus, \( D_{\text{Code}} \), consisting of open-source code extracted from GitHub heads for a variety of programming languages to train \( m_A \).

#### Models

Similar to §4.1, a version of the PaLM2-XXS model has been further pre-trained on \( D_{\text{Code}} \) is used as \( m_A \), while the base pre-trained PaLM2-S model acts as \( m_B \). We build \( m_{A\oplus B} \) by training CALM with only 7% of the same code data (data used for \( m_A \)) to have a data parity.

#### Evaluation Tasks

We evaluate the efficacy of CALM on three different tasks:

- **(i) Code-Completion (CC):** Given an initial set of lines of a code, the model is prompted to complete the code snippet. Here the aim is to evaluate the model for code syntax. We perform zero-shot evaluations on HumanEval benchmark dataset (Chen et al., 2021) and report the Pass@1 (P@1) metric.

- **(ii) Text-to-Code (T2C):** Given a textual context, the model is prompted to generate the corresponding code snippet. Here, the evaluation indicates language understanding and code generation capabilities. We perform 3-shot inference on the MBPP dataset (Austin et al., 2021) and report P@1.

- **(iii) Code-to-Text (C2T):** Given a code snippet, the goal is to generate a natural language explanation of the code. This task evaluates code understanding and text generation. We perform 3-shot evaluations on the CodeXGlue benchmark (Lu et al., 2021) and report chrF1 scores across languages.

#### Results

Table 4 reports comparative performance for the individual models \( m_A \) and \( m_B \), the composed version \( m_{A\oplus B} \), and a fine-tuned anchor baseline \( m_{\text{Code}} \). Firstly, evaluations on the HumanEval dataset suggest that \( m_A \) has a superior understanding of code syntax as a result of its additional training on \( D_{\text{Code}} \). While, due to the larger scale and general purpose pre-training of \( m_B \), it excels at general language understanding and hence performs better on the T2C and C2T tasks.

When employing CALM to compose the two models, we observe a clear transfer and composition of capabilities through significant performance improvements: 6.1% and 3.6% absolute gains over \( m_B \) on the CC and T2C tasks, respectively. We observe that fine-tuning \( m_B \) on \( D_{\text{Code}} \) leads to a significant decline in the C2T performance due to catastrophic forgetting. CALM retains the perfor-
Table 5: Comparative performance of CALM ($m_{\alpha \oplus B}$) across various possible ablations. The metric “#($>m_B$)" depicts the number of languages for which the corresponding model is better than the base for NTL, $m_B$—out of 192, 25, and 11 languages for the three tasks respectively.

4.4 ABLATIONS

Influence of $m_\alpha$ We first study the influence of $m_\alpha$ by replacing it with vanilla and random variants during composition. Table 5 shows the variation of performance across NTL and Code tasks when the specialized $m_\alpha$ is replaced with a vanilla PaLM2-XXS checkpoint or an untrained version of the model, i.e., a random model. We see that there is a considerable drop of performance with these variants across all tasks. On FLORES-200 XX-En task, languages improved with composition drop to 115 and 43 with vanilla and random, respectively. A slight improvement of the vanilla model over $m_B$ indicates that an un-specialized model (with a different training regime than $m_B$) might have orthogonal capabilities leading to an enhanced model. This finding validates that performance gains seen with CALM is a result of utilizing $m_\alpha$ and not just the additional $\Theta_C$ parameters.

Influence of iterative decoding We also investigate a variation where we use $m_\alpha$ as an encoder, i.e., an output token decoded at a given timestep is not amended to $m_\alpha$’s input. In this case, only the prefix representations of $m_\alpha$ are used. This setting eludes to past work for image and text models (Alayrac et al., 2022) where encoder and decoder models are composed. We observe a significant decline in performance across our various tasks when employing this setting.

Comparison with LoRA Finally, we evaluate a parameter efficient fine-tuning approach by training LoRA (Hu et al., 2021) layers to adapt $m_B$. For all experiments, we set the LoRA rank such that the number of added parameters is equal to the number of parameters introduced with CALM. We also train LoRA on the same data as CALM, i.e., $D_C$. We see a considerable difference in performance between the two approaches across all tasks and metrics.

5 CONCLUSION

The proposed CALM framework composes an anchor LLM with specialized augmenting models to enable new tasks not achievable by either models individually. CALM does not require updating the individual models and learns a dense interaction between the models through a few trainable cross-attention parameters. Our experiments present consistent evidence that CALM learns to utilize the expertise from the two models. That is, when composed with relevant augmenting models, we observe a significant uptick in the anchor model’s performance across multiple challenging tasks, such as low-resource translation, reasoning, and code explanation/generation.

That is, CALM is especially useful in scenarios where proprietary data and knowledge is stored in parametric models. With CALM, a foundational LLM could be augmented with such proprietary models to extend a variety of foundational capabilities such as reasoning, world knowledge, and coherent generation over the target proprietary domains. Finally, extensions of CALM could be used to acquire distinct knowledge from multiple augmenting models.
ACKNOWLEDGMENTS

This work was done during RB’s pre-doctoral tenure at Google Research, India (GRI) with PT and PJ. RB is indebted to Manish Gupta, Divvy Thakkar, and all others who enabled this opportunity. RB would also like to thank the members of the Languages team and other researchers at GRI (and beyond), including the incredible pre-doctoral cohort. This work wouldn’t have been possible without their constant support. Namely: Aishwarya P.S., Laurent El Shafey, and Qiao Zhang for their massive help in coding and debugging; Palak Jain and Sagar Gubbi for their feedback and support throughout the project; Kartikeya Badola, Shreyas Havaldar, Amandeep Kaur, and Rishabh Tiwari for being the first ears to all ideas; Cyrus Rashtchian and Richa Dixit for their mentorship.

REFERENCES


Aitor Lewkowycz, Anders Aureassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay V. Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-


A SUPPLEMENTARY MATERIAL FOR NTL

A.1 FLORES-200

Figure 2 depicts the gains over the anchor PaLM2-S model when augmented with a model that has been trained on D_{NTL}. We see a positive gain through CALM for 175 of 192 languages. The highest gains are seen for low-resource languages since they are the most underrepresented in the original model. Diminishing returns with higher resource languages is seen and this trend is similar to the trend seen for m_{NTL}.

![Figure 2: Gains seen by the composed model m_{A\oplus B} over the anchor model, m_B, for the complete set of FLORES-200 languages. The languages are sorted from low to high-resource.]

<table>
<thead>
<tr>
<th>Language</th>
<th>mA</th>
<th>mB</th>
<th>mA\oplus B (CALM)</th>
<th>m_B^{NTL}</th>
</tr>
</thead>
<tbody>
<tr>
<td>meo</td>
<td>7.6</td>
<td>28.8</td>
<td>34.0</td>
<td>33.2</td>
</tr>
<tr>
<td>mfa</td>
<td>4.0</td>
<td>14.0</td>
<td>17.6</td>
<td>20.4</td>
</tr>
<tr>
<td>pcm</td>
<td>4.4</td>
<td><strong>34.4</strong></td>
<td>33.6</td>
<td>31.6</td>
</tr>
<tr>
<td>efi</td>
<td>3.2</td>
<td>14.8</td>
<td>18.0</td>
<td>14.0</td>
</tr>
<tr>
<td>min</td>
<td>6.0</td>
<td><strong>25.2</strong></td>
<td>23.6</td>
<td>24.8</td>
</tr>
<tr>
<td>ilo</td>
<td>4.8</td>
<td>14.8</td>
<td><strong>16.8</strong></td>
<td>14.0</td>
</tr>
<tr>
<td>ady</td>
<td>6.4</td>
<td>30.0</td>
<td><strong>36.4</strong></td>
<td>29.2</td>
</tr>
<tr>
<td>mai</td>
<td>3.2</td>
<td><strong>22.8</strong></td>
<td>24.8</td>
<td>21.2</td>
</tr>
<tr>
<td>nso</td>
<td>6.0</td>
<td><strong>8.4</strong></td>
<td><strong>8.4</strong></td>
<td>9.6</td>
</tr>
<tr>
<td>mzn</td>
<td>4.8</td>
<td>31.6</td>
<td><strong>36.4</strong></td>
<td>27.6</td>
</tr>
<tr>
<td>bew</td>
<td>4.4</td>
<td><strong>33.6</strong></td>
<td><strong>34.8</strong></td>
<td>33.6</td>
</tr>
<tr>
<td>ts</td>
<td>4.8</td>
<td>7.2</td>
<td><strong>10.0</strong></td>
<td>11.6</td>
</tr>
<tr>
<td>dv</td>
<td>2.8</td>
<td>11.2</td>
<td><strong>14.8</strong></td>
<td>13.2</td>
</tr>
<tr>
<td>avg.</td>
<td><strong>4.5</strong></td>
<td><strong>18.6</strong></td>
<td><strong>21.4</strong></td>
<td>19.8</td>
</tr>
</tbody>
</table>

Table 6: Performance evaluations on the complete set of low-resource languages for GSM-8K. Augmenting m_A with m_B as m_{A\oplus B} improves performance over m_B across a majority of languages. On average, we see an improvement of 2.8%.
Table 7: Quality evaluation for the LRL GSM-8K dataset across languages. We created the dataset by translating the original English sentences of GSM-8K to the target language using the Google Translate API. We measure quality by back-translating the obtained examples back to English and measuring: (i) The overlap between the back-translated and the original English sentence, and (ii) The delta change in performance when PaLM2-S is evaluated on this back-translated version of GSM-8K as compared to the original version.

### A.2 GSM-8K

**Quality evaluation for LRL GSM-8K**

As described in Section 4.2, we created the GSM-8K dataset (Cobbe et al., 2021) for low-resource languages by using the Google Translate API to obtain silver translations in the target language from the source English sentence in the original dataset. We perform a quality evaluation of these examples by back-translating them back to English using the same translation API and defining two metrics over it:

1. **Overlap**: The BLUE score measure between the actual example and the back-translated example,
2. **Delta**: The change in performance of the PaLM2-S model when evaluated on the original GSM-8K set as compared to the back-translated version.

Table 7 shows the values for these metrics across the various languages. We see that a decently high overlap value is seen across all languages. At the same time, the delta in performance is also minimal indicating that key attributes in the GSM-8K examples are not affected by translation.

**Results on the complete language set**

Table 6 shows the comparative evaluations on the complete set of 25 low-resource languages for which GSM evaluations are performed. We see an improvement over the anchor model $m_B$ for 20 of 25 languages. We also compare against the fully continued pretrained version $m_B^{NL}$ and observe that $m_{A\oplus B}$ outperform it for 18 of 25 languages.

### B Qualitative Analysis

Table 8 depicts a few qualitative examples for the code-to-text, or the code explanation task, for Python. These examples depict examples for the three broader bucket of examples that we observe in cases when CALM yields the correct responses:

1. When neither of $m_A$ or $m_B$ generates the correct response but $m_{A\oplus B}$ correctly attends over their latent representations to yield the correct output,
2. When either of $m_A$ or $m_B$ is seen to give the correct response while the other one is incorrect and $m_{A\oplus B}$ generates the correct response that matches the generation from the correct model of $m_A$ and $m_B$, and
3. When both $m_A$ and $m_B$ generate the correct response and $m_{A\oplus B}$ reproduces those generations.

We also observed similar qualitative patterns with other tasks for language inclusivity.

### C Overhead with CALM

In this section, we include a detailed computation of the expected parametric and training overhead while composing given models using our proposed CALM framework.
```python
def ConsumeBool(self):
    try:
        result = ParseBool(self.token)
    except ValueError as e:
        raise self.ParseError(str(e))
    self.NextToken()
    return result

⇒ Consumes a boolean
m_A: Consumes a boolean
m_B: The object is not a member
CALM: Consumes a boolean

def value(self):
    if self.has_value:
        return self.impl[OBJ].get_val(K)
    else:
        raise ValueError("Not found")
    return

⇒ Print an error message and exit.
[a part of the given model prefix]
Exit with error message
Print an error message and exit

def get_positions(url):
    data = _get_resource(url)
    positions = [x for x in data[‘p’]]
    return positions

⇒ Returns a list of positions.
Positions of specified instruments.
Get all positions.
Returns a list of positions.

def distance(x0, y0, x1, y1):
    return (
        sqrt(pow(x1-x0,2) + pow(y1-y0,2))
    )

⇒ Returns the distance between two points
Calculates the distance between two points
Return the distance between two points
Calculates the distance between two points
```

Table 8: Cherry-picked qualitative examples for the code-to-text task on Python that depict examples that fall into a set of larger bucket of patterns that we observe across examples. CALM does well in various settings: (i) when \( m_A \) produces the correct output but not \( m_B \), (ii) vice-versa—when \( m_B \) does well, and (iii) when neither of the two base models do well but a combination of intermediate representations allow the composed model to give the correct output. This shows that composition implicitly learns to do both: routing across models and a combination, based on a given input.
C.1 Parametric Overhead

Building from the notations in §3.1, let’s say the two models \( m_A \) and \( m_B \) have \( N_A \) and \( N_B \) number of standard transformer layers, respectively, with each layer of output dimensionality \( D_A \) and \( D_B \). As mentioned, we choose \( n = n_A = n_B \) number of layers to perform the composition.

\[
\begin{align*}
\# \text{params per } f_{\text{proj}} \text{ layer} &= (D_A * D_B) \\
\# \text{params per } f_{\text{cross}} \text{ layer} &= (3 * D_B^2) \\
\# \text{total added params} &= n * (D_A * D_B + 3 * D_B^2) \\
\# \text{params in } m_B &= N_B * (V_B * D_B + 3 * D_B^2 + 2 * D_B * D_B * K_B) \\
\% \text{age added parameters} &= (15M/1B) * 100 = 1.5%
\end{align*}
\]

where, \( V_B \) and \( K_B \) depict the vocabulary size and hidden multiplication factor, respectively.

Let’s consider some standard transformer configurations to understand the parameter overhead. As an example, consider the layer configurations of standard BERT models: BERT-small (\( m_A \)) and BERT-large (\( m_B \)). That is: \( N_A = 4, D_A = 512, N_B = 24, D_B = 1024, V_B = 30K, K_B = 4 \). Assuming that we select all layers of \( m_B \), the value of \( n = 4 \). Then:

\[
\begin{align*}
\# \text{params for CALM} &= 4 * (512 * 1024) \\
&= 24 * (30K * 1024) \\
&= 24 * (30K * 1024) \\
&= (15M/1B) * 100 = 1.5%
\end{align*}
\]

Hence, number of parameters added during composition is approximately 1.5% of those in \( m_B \).

C.2 Training Overhead

Although learning the new parameters during CALM training requires back propagating over \( m_B \), the total training costs are still significantly lesser than fine-tuning \( m_B \), owing to the significantly lesser training examples.

Firstly, as discussed in the previous section, the additional number of parameters introduced during composition is \(~1.5\%\) of those in \( m_B \)—hence, a negligible parametric addition. Further, since only \(~5-7\%\) of the total \( m_B \) fine-tuning data is required to train CALM, the training cost of CALM is minimal with respect to that of fine-tuning \( m_B \).

Let’s assume that (i) the number of parameters in \( m_B \) is \( P_B \) and the number of examples required to fine-tune \( m_B \) is \( D_B \), (ii) the cost (in FLOPS), \( C(\cdot) \), of training scales linearly with parameters and data, i.e., \( C(m_B) = O(P_B * D_B) \) (iii) the number of parameters in \( m_A \) is 10% of those in \( m_B \), (iv) the number of parameters added for CALM is 2% of those on \( m_B \), and (v) the amount of data required to train CALM is 5% of \( m_B \) training. Then:

\[
\begin{align*}
P_{m_{A:B}} &= P_A + P_B + P_{\text{in}} \\
&= 0.10 * P_B + P_B + 0.02 * P_B \\
&= 1.12 * P_B \\
D_{m_{A:B}} &= 0.05 * D_B \\
C(m_{A:B}) &= O(P_{m_{A:B}} * D_{m_{A:B}}) \\
&= O((1.12 * P_B) * (0.05 * D_B)) \\
&= O(0.056 * P_B * D_B) \\
&= 5.6\% * O(P_B * D_B) \\
&= 5.6\% * C(m_B)
\end{align*}
\]

Hence, the cost of training CALM is a significantly lesser (<10%) than that of fine-tuning \( m_B \).