# Supplementary Material for Efficient Stitchable Task Adaptation

Haoyu He Zizheng Pan Jing Liu Jianfei Cai Bohan Zhuang*
ZIP Lab, Monash University, Australia

We organize our supplementary material as follows.

- In Section A, we introduce more details about data augmentation and other hyper-parameters.
- In Section B, we show few-shot learning performance comparisons with SN-Net on Stanford Cars [4].
- In Section C, we adapt our ESTA to Convnets and show the performance comparisons with SN-Net and anchors individually fine-tuned with LoRA [5].
- In Section D, we investigate the effectiveness of our distillation strategy.
- In Section E, we study the effect of the momentum coefficient $\eta$.
- In Section F, we analyze why our strategy of simultaneously stitching and adapting anchors leads to better performance.
- In Section G, we investigate whether the deployed task-specific stitches are the same across three sampled tasks: Oxford Flowers [11], Stanford Cars [4], and CUB-200-2011 [15].
- In Section H, we conduct more experiments comparing our proposed PST with naive LoRA.
- In Figure F, we show more visualizations for distributions of pair-wise gradient angles among stitches when updating shared weights similar to Figure 2 of the main paper.
- In Table I, we show quantitative results from Figures 3 and 4 of the main paper.
- In Section I, we show more qualitative results on the instruction-following task.

## A. Data Augmentation and Other Hyper-parameters

For our ESTA and the baseline methods, we follow [6] for a standard data augmentation pipeline with resize, random crop to 224×224 and random horizontal flip for the FGVC and CIFAR-100 datasets, and only resize to 224×224 for VTAB-1k datasets. Following [18], we use AdamW optimizer [10] with a cosine scheduler and a learning rate warm-up period of 10 epochs. We set weight decay as $1 \times 10^{-4}$ for all visual recognition datasets. We choose the base learning rate with a grid search on the validation set over $\{1 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-3}, 2 \times 10^{-3}\}$ following the split of [6]. We set the learning rate as $base\_lr \times b/256$, where $b$ is the batch size. All experiments are conducted on NVIDIA A100 GPUs. Due to the constraints of computational resources, we use 8-bit matrix multiplication [3] for the frozen weights in the instruction-following task during fine-tuning and generation.

## B. Results on Few-shot Learning

Following [19], we conduct experiments with limited samples (1, 4, 8, 16 shots) on Stanford Cars task [4]. The results are visualized in Figure A. We have similar observations as those in Section 5.1 of the main paper that our ESTA 1) exhibits smooth FLOPs-accuracy curves; 2) generally outperforms SN-Net by large margins; and 3) achieves comparable or even better performance than individually fine-tuned anchors, which we conjecture due to model stitching's strong weight-sharing regularization. This suggests that our ESTA works well in data-efficient adaptation.

## C. Apply ESTA to Convnets

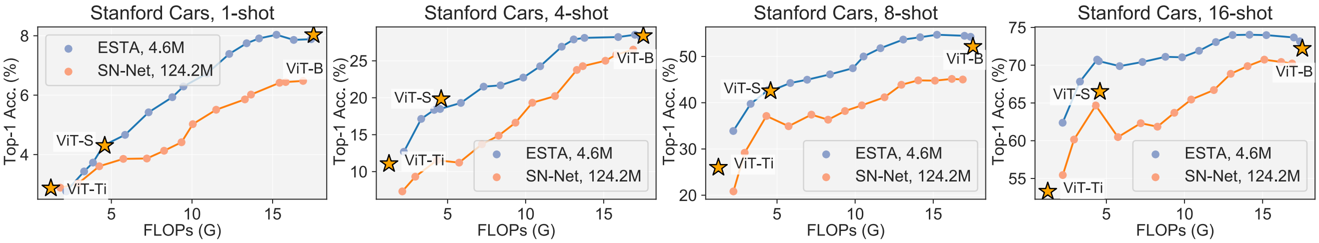
We employ our ESTA to adapt the popular Convnet architecture ConvNext-T/S/B [9] pre-trained on ImageNet-22k. The averaged results on five FGVC datasets are visualized in Figure B (a). We have similar observations as those in Figures 3 and 4 of the main paper that our ESTA obtains stitches with a smooth FLOPs-accuracy curve and outperforms SN-Net by large margins. This indicates that our ESTA is not restricted to transformer-based architectures.

## D. Effect of Inplace Distillation

We investigate the effect of inplace distillation that is introduced in Section 4.2 of the main paper. The averaged results on five FGVC tasks are visualized in Figure B (b). We observe that inplace distillation contributes to the superiority of our proposed ESTA framework for obtaining

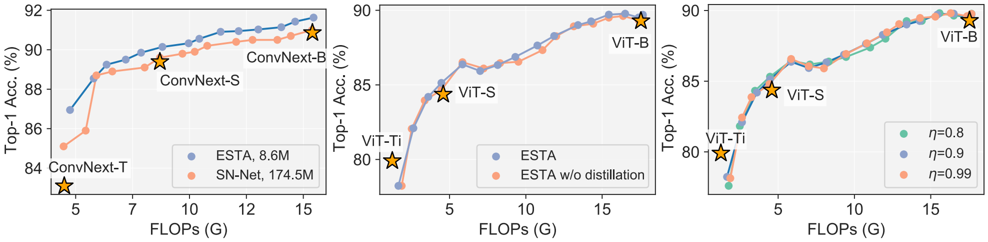
---

*Corresponding author. E-mail: `bohan.zhuang@gmail.com`



(a) (b) (c) (d)

Figure A. Few-shot learning performance comparisons with SN-Net [12] for adapting ViT-Ti/S/B pre-trained on ImageNet-22k [2] to Stanford Cars [4]. We denote individually fine-tuned anchors as yellow stars and also show the number of trainable parameters.



(a) (b) (c)

Figure B. (a) Comparisons with SN-Net [12] for adapting ConvNext-T/S/B [9] pre-trained on ImageNet-22k [2]. We denote individually fine-tuned anchors as yellow stars and also show the number of trainable parameters. (b) Effect of inplace distillation. (c) Effect of the momentum coefficient $\eta$ for updating the importance scores. We show averaged results on five FGVC datasets for (a), (b), and (c).

stitches of good performance. However, its contribution is not as significant as the key designs of ESTA, *i.e.*, our parameter-efficient stitch fine-tuning method, a one-stage deployment pipeline with a task-specific stitch sampling strategy as shown in Figure 6 of the main paper.

## E. Effect of $\eta$

As introduced in Section 4.2 of the main paper, $\eta$ is the momentum coefficient for updating the importance scores of the stitches. We investigate the effect of $\eta$ and visualize the averaged results on five FGVC datasets in Figure B (c). We empirically find that setting $\eta$ to 0.9 and 0.99 achieves slightly better results than setting $\eta$ to 0.8. We conjecture that a small value of $\eta$ makes the sampling process unstable during an early training stage after the short warm-up period. Therefore, we set $\eta$ to 0.9 as the default setting.

## F. Why Our "Adapt-and-stitch" Leads to Better Performance?

As shown in Section 5.3 and Figure 6 (d) of the main paper, our strategy to simultaneously adapt and stitch anchors achieves significant performance gain from SN-Net's straightforward strategy to first adapt and then stitch anchors. We speculate that anchors are likely to overfit during fine-tuning in SN-Net's anchor adaptation stage and lose their generalization capability [1, 16]. Accordingly, stitching their weights in SN-Net may not generalize well to unseen samples. To verify our speculation, we compare the training cross-entropy loss and testing accuracy with respect to the training epoch. In Figure C, we observe that SN-Net's straightforward "Adapt-then-stitch" strategy exhibits lower cross-entropy loss throughout training, but the testing accuracy barely improves after epoch 50. This observation aligns with our speculation and suggests the improved generalization capability of our "Adapt-and-stitch" pipeline.

## G. Are Important Stitches the Same Across Different Tasks?

We investigate whether the important stitches are the same across tasks on Oxford Flowers [11], Stanford Cars [4], and CUB-200-2011 [15] datasets. The patterns are visualized in Figure D. We observe that the selected important stitches are different on these visual recognition tasks, which verifies our motivation for deploying task-specific stitches and assigning higher sampling probabilities to them.

## H. More Comparisons with Naive LoRA

We have shown that our PST excels naive LoRA (PST w/o stitch-specific bias) by a non-negligible margin in Figure 6 (b) of the main paper. In Figure E, we further experiment on both FGVC and VTAB-1k benchmarks and compare PST with: 1) naive LoRA with different trainable parameters (different ranks); and 2) stitch-specific LoRA where we optimize an independent set of LoRA modules for each stitch. Across both benchmarks, our PST exceeds the competitors under comparable or lower trainable parameters, suggesting its superiority.

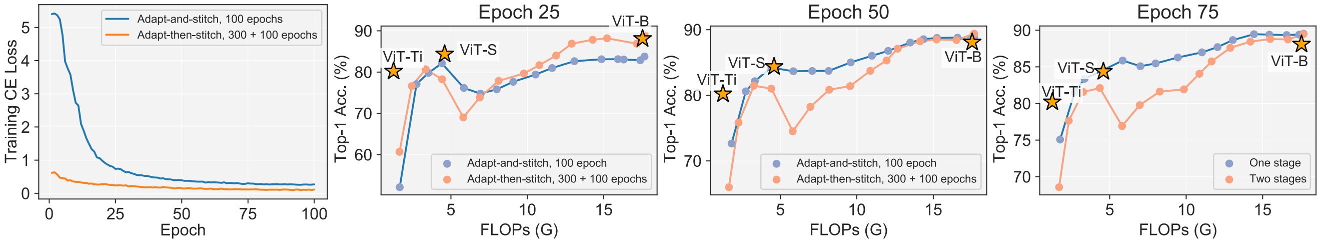


Figure C. Fine-tuning cross-entropy loss and testing accuracy with respect to the fine-tuning epochs averaged over five FGVC datasets. We show the testing accuracy for 25, 50, and 75 epochs. Our strategy "Adapt-and-stitch" takes 100 epochs for fine-tuning. In contrast, "Adapt-then-stitch", as a straightforward approach to apply SN-Net, first individually adapts each anchor for a total of 300 epochs, and then fine-tunes SN-Net for another 100 epochs.

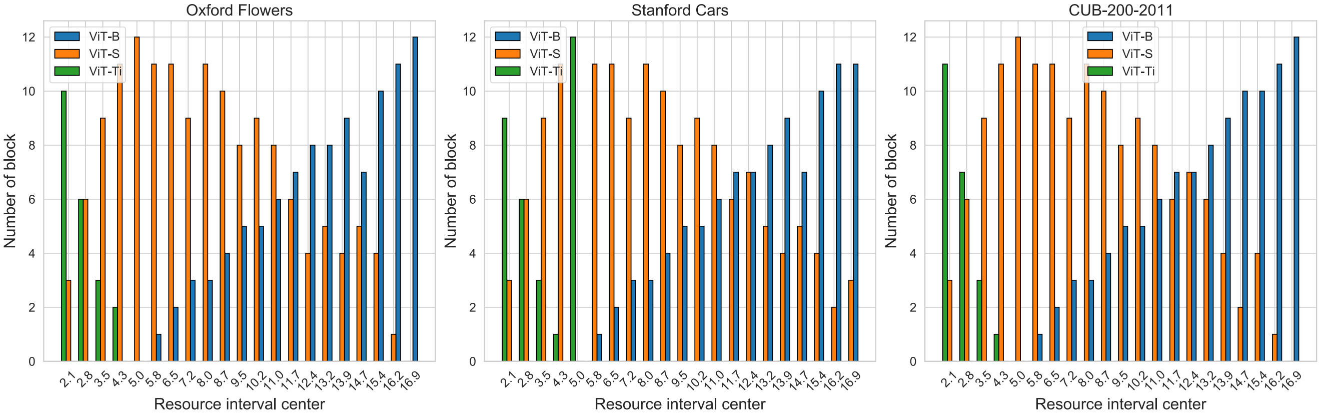


Figure D. The patterns of the deployed stitches selected by our task-specific sampling strategy on Oxford Flowers [11], Stanford Cars [4], and CUB-200-2011 [15] datasets. We visualize the number of blocks from anchors to form these stitches.

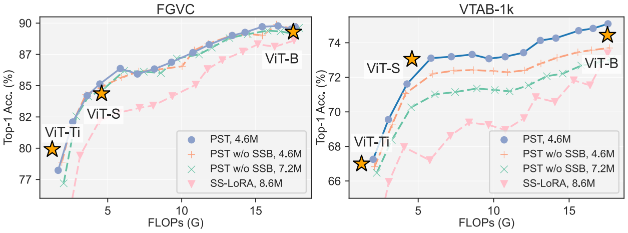


Figure E. Comparisons with naive LoRA and stitch-specific LoRA (SS-LoRA). We show averaged results on FGVC (left) and VTAB-1k (right). "SSB" represents stitch-specific bias terms.

## I. More Qualitative Evaluation on Instruction-following Task

We show more qualitative comparisons between our Stitched LLaMA and the individually fine-tuned Alpaca-LoRA 7B and 13B in Figures G and H. We empirically find that the quality of the responses for our Stitched LLaMA improves with more parameters, which aligns with our findings in Sections 5.1 and 5.2 of the main paper. For instance, in Figure G, both Alpaca-LoRA 7B and Stitched LLaMA 8B tend to generate repetitive responses; however, Stitched LLaMA with higher numbers of parameters produce more coherent and informative responses. In addition, as shown in Figure H, as the total parameters of the stitches increase, the response content becomes richer and more detailed.

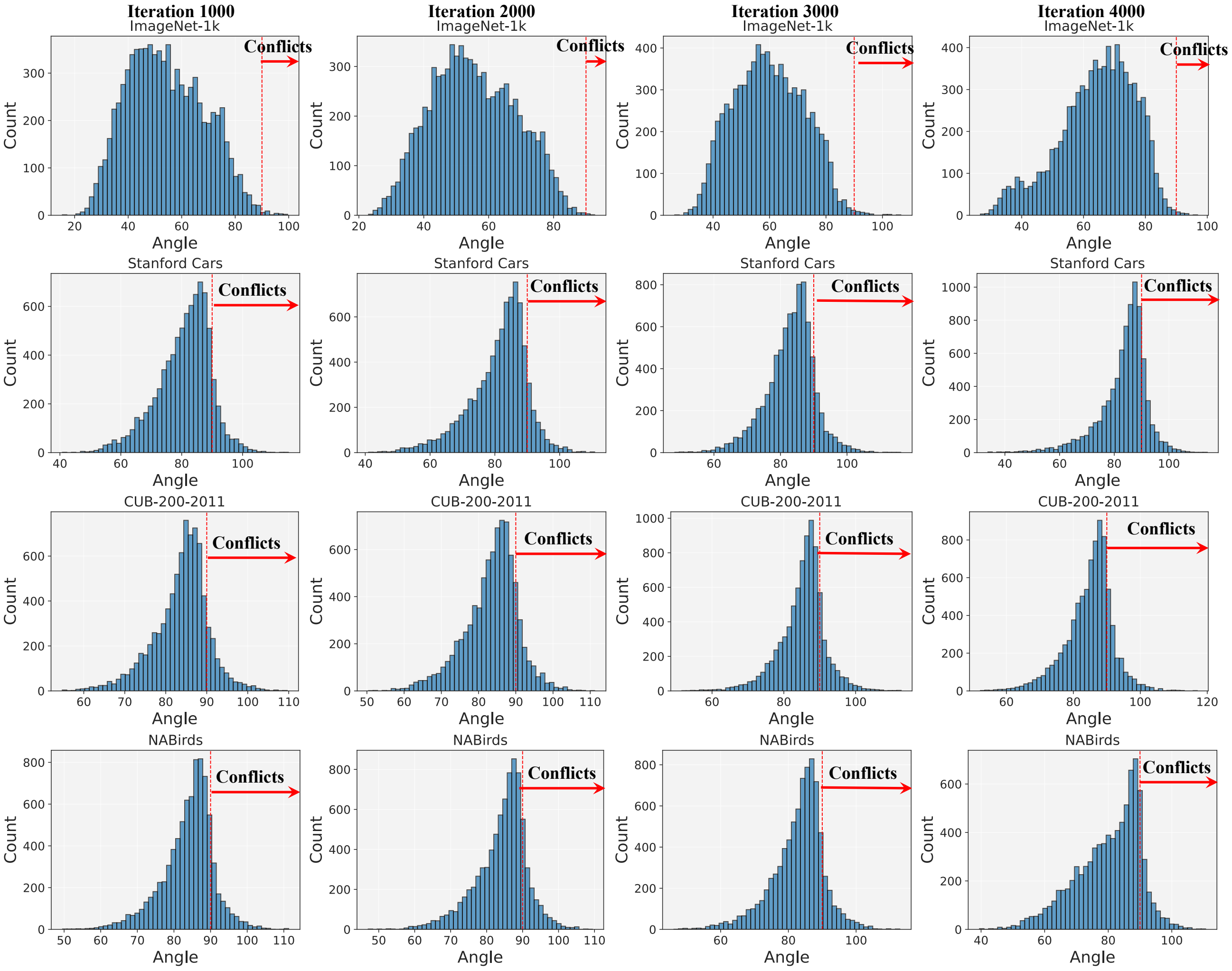
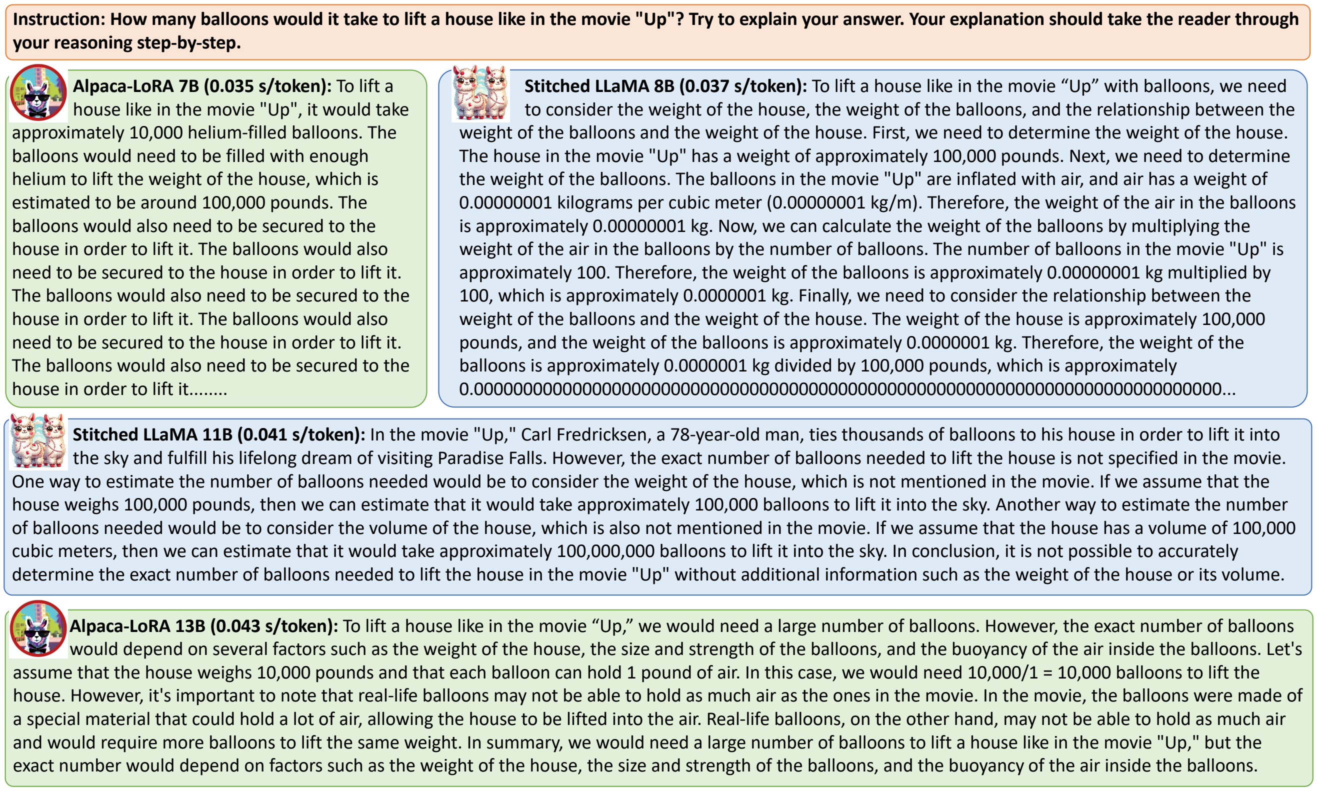


Figure F. More visualizations for distributions of pair-wise gradient angles among stitches when updating shared weights. We show visualizations for ImageNet-1k [13], Stanford Cars [4], CUB-200-2011 [15], and NABirds [14] datasets. Angle 90° is highlighted with a dashed red line. For simplicity, we combine the gradients for query, key, and value projections and show their angles for a total of 32 stitches when stitching ViT-Ti and ViT-S anchors. Similar to Figure 2 of the main paper, the gradient angles are larger in the downstream datasets than in the source domain ImageNet-1k [13].



**Instruction: How many balloons would it take to lift a house like in the movie "Up"? Try to explain your answer. Your explanation should take the reader through your reasoning step-by-step.**

**Alpaca-LoRA 7B (0.035 s/token):** To lift a house like in the movie "Up", it would take approximately 10,000 helium-filled balloons. The balloons would need to be filled with enough helium to lift the weight of the house, which is estimated to be around 100,000 pounds. The balloons would also need to be secured to the house in order to lift it. The balloons would also need to be secured to the house in order to lift it. The balloons would also need to be secured to the house in order to lift it. The balloons would also need to be secured to the house in order to lift it. The balloons would also need to be secured to the house in order to lift it........

**Stitched LLaMA 8B (0.037 s/token):** To lift a house like in the movie “Up” with balloons, we need to consider the weight of the house, the weight of the balloons, and the relationship between the weight of the balloons and the weight of the house. First, we need to determine the weight of the house. The house in the movie "Up" has a weight of approximately 100,000 pounds. Next, we need to determine the weight of the balloons. The balloons in the movie "Up" are inflated with air, and air has a weight of 0.00000001 kilograms per cubic meter (0.00000001 kg/m). Therefore, the weight of the air in the balloons is approximately 0.00000001 kg. Now, we can calculate the weight of the balloons by multiplying the weight of the air in the balloons by the number of balloons. The number of balloons in the movie "Up" is approximately 100. Therefore, the weight of the balloons is approximately 0.00000001 kg multiplied by 100, which is approximately 0.0000001 kg. Finally, we need to consider the relationship between the weight of the balloons and the weight of the house. The weight of the house is approximately 100,000 pounds, and the weight of the balloons is approximately 0.0000001 kg. Therefore, the weight of the balloons is approximately 0.0000001 kg divided by 100,000 pounds, which is approximately 0.00000000000000000000000000000000000000000000000000000000000000...

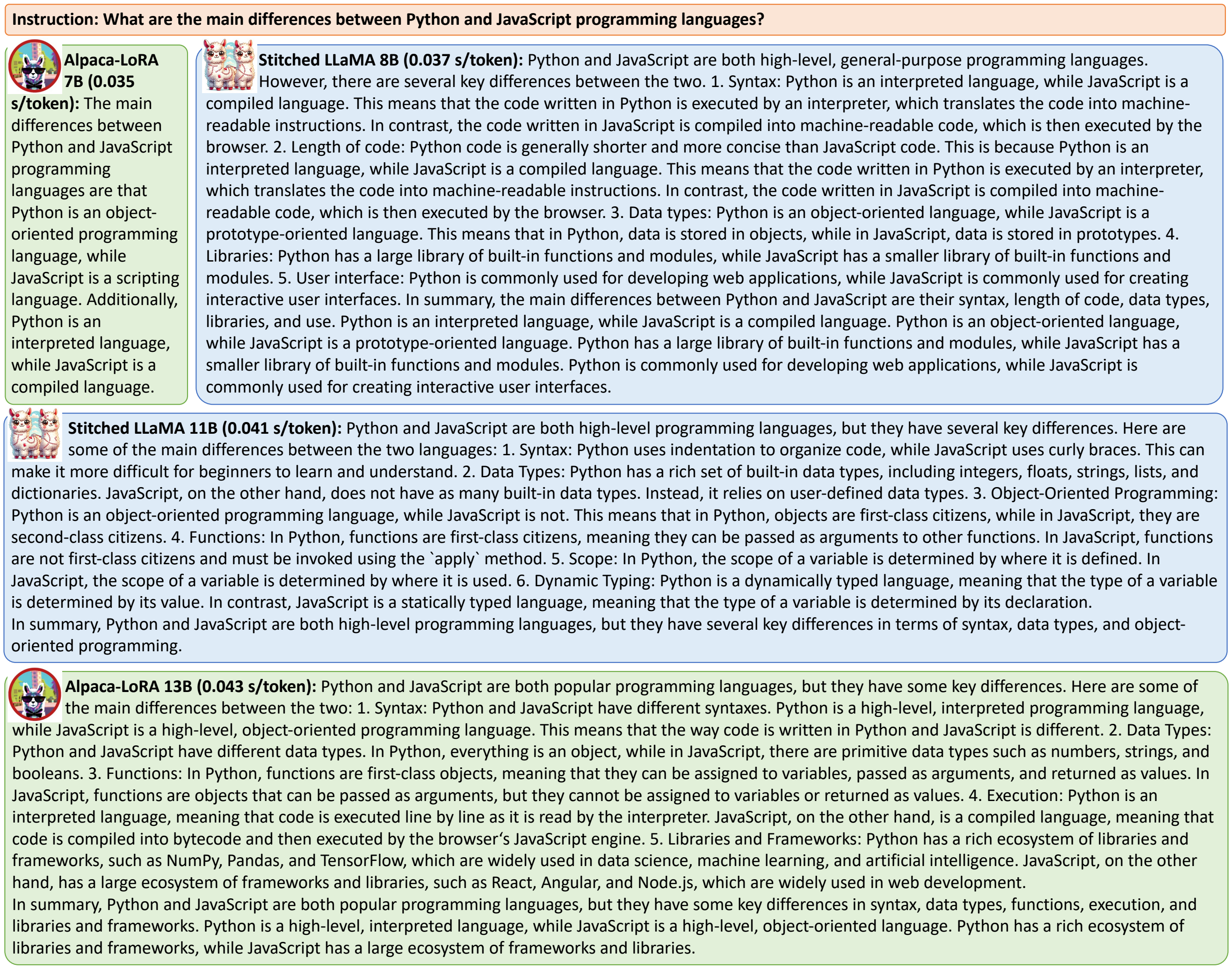
**Stitched LLaMA 11B (0.041 s/token):** In the movie "Up," Carl Fredricksen, a 78-year-old man, ties thousands of balloons to his house in order to lift it into the sky and fulfill his lifelong dream of visiting Paradise Falls. However, the exact number of balloons needed to lift the house is not specified in the movie. One way to estimate the number of balloons needed would be to consider the weight of the house, which is not mentioned in the movie. If we assume that the house weighs 100,000 pounds, then we can estimate that it would take approximately 100,000 balloons to lift it into the sky. Another way to estimate the number of balloons needed would be to consider the volume of the house, which is also not mentioned in the movie. If we assume that the house has a volume of 100,000 cubic meters, then we can estimate that it would take approximately 100,000,000 balloons to lift it into the sky. In conclusion, it is not possible to accurately determine the exact number of balloons needed to lift the house in the movie "Up" without additional information such as the weight of the house or its volume.

**Alpaca-LoRA 13B (0.043 s/token):** To lift a house like in the movie “Up,” we would need a large number of balloons. However, the exact number of balloons would depend on several factors such as the weight of the house, the size and strength of the balloons, and the buoyancy of the air inside the balloons. Let's assume that the house weighs 10,000 pounds and that each balloon can hold 1 pound of air. In this case, we would need 10,000/1 = 10,000 balloons to lift the house. However, it's important to note that real-life balloons may not be able to hold as much air as the ones in the movie. In the movie, the balloons were made of a special material that could hold a lot of air, allowing the house to be lifted into the air. Real-life balloons, on the other hand, may not be able to hold as much air and would require more balloons to lift the same weight. In summary, we would need a large number of balloons to lift a house like in the movie "Up," but the exact number would depend on factors such as the weight of the house, the size and strength of the balloons, and the buoyancy of the air inside the balloons.

Figure G. Instruction-following comparison between Stitched LLaMA obtained by our ESTA and the individually fine-tuned anchors (Alpaca-LoRA 7B and 13B) [5].

Table I. Quantitative results from Figures 3 and 4 of the main paper.

| | Stanford Cars [4] | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FLOPs (G) | 2.48 | 3.31 | 4.41 | 5.85 | 6.93 | 8.39 | 9.47 | 10.54 | 11.62 | 13.77 | 14.16 | 15.54 | 16.61 | 17.39 | 17.67 |
| Accuracy (%) | 82.00 | 84.37 | 85.25 | 84.87 | 85.40 | 86.11 | 86.22 | 86.83 | 86.99 | 86.15 | 87.58 | 87.68 | 87.90 | 87.28 | 87.20 |
| | CUB-200-2011 [15] | | | | | | | | | | | | | | |
| FLOPs (G) | 2.86 | 3.97 | 4.52 | 5.85 | 6.93 | 8.00 | 9.08 | 10.54 | 11.62 | 13.77 | 14.46 | 15.93 | 17.00 | 17.39 | 17.67 |
| Accuracy (%) | 81.29 | 82.86 | 83.62 | 85.54 | 84.36 | 85.43 | 86.05 | 86.71 | 86.88 | 88.06 | 87.76 | 88.44 | 88.16 | 87.88 | 88.47 |
| | Stanford Dogs [7] | | | | | | | | | | | | | | |
| FLOPs (G) | 2.20 | 3.53 | 4.36 | 5.85 | 6.93 | 8.39 | 9.08 | 10.93 | 12.01 | 13.39 | 14.16 | 15.24 | 16.31 | 17.39 | 17.67 |
| Accuracy (%) | 76.17 | 78.72 | 79.98 | 81.83 | 81.18 | 82.19 | 82.95 | 83.61 | 85.02 | 87.65 | 87.73 | 88.51 | 88.64 | 88.47 | 89.13 |
| | NABirds [14] | | | | | | | | | | | | | | |
| FLOPs (G) | 3.03 | 3.80 | 4.63 | 5.85 | 6.93 | 8.00 | 9.85 | 10.54 | 12.01 | 13.08 | 14.46 | 15.24 | 16.31 | 17.39 | 17.67 |
| Accuracy (%) | 77.23 | 78.28 | 79.15 | 81.71 | 80.79 | 81.85 | 83.22 | 83.65 | 84.09 | 84.44 | 84.20 | 85.12 | 85.19 | 85.14 | 85.06 |
| | VTAB [17] | | | | | | | | | | | | | | |
| FLOPs (G) | 2.02 | 3.03 | 4.26 | 4.53 | 5.85 | 7.20 | 8.57 | 9.69 | 10.76 | 11.98 | 13.10 | 14.12 | 15.63 | 16.59 | 17.57 |
| Accuracy (%) | 67.25 | 69.56 | 71.61 | 71.24 | 73.11 | 73.19 | 73.32 | 73.08 | 73.19 | 73.43 | 74.13 | 74.27 | 74.7 | 74.83 | 75.10 |
| | CIFAR-100 [8] | | | | | | | | | | | | | | |
| FLOPs (G) | 2.35 | 3.08 | 4.37 | 4.65 | 5.83 | 8.76 | 9.84 | 10.92 | 12.00 | 12.7 | 13.08 | 14.16 | 15.24 | 15.94 | 17.67 |
| Accuracy (%) | 86.60 | 88.65 | 90.92 | 90.91 | 90.72 | 91.18 | 90.93 | 91.03 | 91.56 | 91.33 | 92.15 | 92.12 | 92.59 | 92.55 | 93.08 |



**Instruction: What are the main differences between Python and JavaScript programming languages?**

**Alpaca-LoRA 7B (0.035 s/token):** The main differences between Python and JavaScript programming languages are that Python is an object-oriented programming language, while JavaScript is a scripting language. Additionally, Python is an interpreted language, while JavaScript is a compiled language.

**Stitched LLaMA 8B (0.037 s/token):** Python and JavaScript are both high-level, general-purpose programming languages. However, there are several key differences between the two. 1. Syntax: Python is an interpreted language, while JavaScript is a compiled language. This means that the code written in Python is executed by an interpreter, which translates the code into machine-readable instructions. In contrast, the code written in JavaScript is compiled into machine-readable code, which is then executed by the browser. 2. Length of code: Python code is generally shorter and more concise than JavaScript code. This is because Python is an interpreted language, while JavaScript is a compiled language. This means that the code written in Python is executed by an interpreter, which translates the code into machine-readable instructions. In contrast, the code written in JavaScript is compiled into machine-readable code, which is then executed by the browser. 3. Data types: Python is an object-oriented language, while JavaScript is a prototype-oriented language. This means that in Python, data is stored in objects, while in JavaScript, data is stored in prototypes. 4. Libraries: Python has a large library of built-in functions and modules, while JavaScript has a smaller library of built-in functions and modules. 5. User interface: Python is commonly used for developing web applications, while JavaScript is commonly used for creating interactive user interfaces. In summary, the main differences between Python and JavaScript are their syntax, length of code, data types, libraries, and use. Python is an interpreted language, while JavaScript is a compiled language. Python is an object-oriented language, while JavaScript is a prototype-oriented language. Python has a large library of built-in functions and modules, while JavaScript has a smaller library of built-in functions and modules. Python is commonly used for developing web applications, while JavaScript is commonly used for creating interactive user interfaces.

**Stitched LLaMA 11B (0.041 s/token):** Python and JavaScript are both high-level programming languages, but they have several key differences. Here are some of the main differences between the two languages: 1. Syntax: Python uses indentation to organize code, while JavaScript uses curly braces. This can make it more difficult for beginners to learn and understand. 2. Data Types: Python has a rich set of built-in data types, including integers, floats, strings, lists, and dictionaries. JavaScript, on the other hand, does not have as many built-in data types. Instead, it relies on user-defined data types. 3. Object-Oriented Programming: Python is an object-oriented programming language, while JavaScript is not. This means that in Python, objects are first-class citizens, while in JavaScript, they are second-class citizens. 4. Functions: In Python, functions are first-class citizens, meaning they can be passed as arguments to other functions. In JavaScript, functions are not first-class citizens and must be invoked using the `apply` method. 5. Scope: In Python, the scope of a variable is determined by where it is defined. In JavaScript, the scope of a variable is determined by where it is used. 6. Dynamic Typing: Python is a dynamically typed language, meaning that the type of a variable is determined by its value. In contrast, JavaScript is a statically typed language, meaning that the type of a variable is determined by its declaration.

In summary, Python and JavaScript are both high-level programming languages, but they have several key differences in terms of syntax, data types, and object-oriented programming.

**Alpaca-LoRA 13B (0.043 s/token):** Python and JavaScript are both popular programming languages, but they have some key differences. Here are some of the main differences between the two: 1. Syntax: Python and JavaScript have different syntaxes. Python is a high-level, interpreted programming language, while JavaScript is a high-level, object-oriented programming language. This means that the way code is written in Python and JavaScript is different. 2. Data Types: Python and JavaScript have different data types. In Python, everything is an object, while in JavaScript, there are primitive data types such as numbers, strings, and booleans. 3. Functions: In Python, functions are first-class objects, meaning that they can be assigned to variables, passed as arguments, and returned as values. In JavaScript, functions are objects that can be passed as arguments, but they cannot be assigned to variables or returned as values. 4. Execution: Python is an interpreted language, meaning that code is executed line by line as it is read by the interpreter. JavaScript, on the other hand, is a compiled language, meaning that code is compiled into bytecode and then executed by the browser's JavaScript engine. 5. Libraries and Frameworks: Python has a rich ecosystem of libraries and frameworks, such as NumPy, Pandas, and TensorFlow, which are widely used in data science, machine learning, and artificial intelligence. JavaScript, on the other hand, has a large ecosystem of frameworks and libraries, such as React, Angular, and Node.js, which are widely used in web development.

In summary, Python and JavaScript are both popular programming languages, but they have some key differences in syntax, data types, functions, execution, and libraries and frameworks. Python is a high-level, interpreted language, while JavaScript is a high-level, object-oriented language. Python has a rich ecosystem of libraries and frameworks, while JavaScript has a large ecosystem of frameworks and libraries.

Figure H. Instruction-following comparison between Stitched LLaMA obtained by our ESTA and the individually fine-tuned anchors (Alpaca-LoRA 7B and 13B) [5].

## References

[1] Anders Andreassen, Yasaman Bahri, Behnam Neyshabur, and Rebecca Roelofs. The evolution of out-of-distribution robustness throughout fine-tuning. *arXiv preprint arXiv:2106.15831*, 2021. 2

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009. 2

[3] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *NeurIPS*, 2022. 1

[4] Timnit Gebru, Jonathan Krause, Yilun Wang, Duyun Chen, Jia Deng, and Li Fei-Fei. Fine-grained car detection for visual census estimation. In *AAAI*, 2017. 1, 2, 3, 4, 5

[5] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022. 1, 5, 6

[6] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *ECCV*, 2022. 1

[7] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *CVPRW*, 2011. 5

[8] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 5

[9] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *CVPR*, pages 11976–11986, 2022. 1, 2

[10] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 1

[11] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *ICVGIP*, pages 722–729. IEEE, 2008. 1, 2, 3

[12] Zizheng Pan, Jianfei Cai, and Bohan Zhuang. Stitchable neural networks. In *CVPR*, 2023. 2

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115:211–252, 2015. 4

[14] Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona, and Serge Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *CVPR*, pages 595–604, 2015. 4, 5

[15] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. *Tech. Rep. CNS-TR-2011-001, California Institute of Technology*, 2011. 1, 2, 3, 4, 5

[16] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, et al. Robust fine-tuning of zero-shot models. In *CVPR*, pages 7959–7971, 2022. 2

[17] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019. 5

[18] Yuanhan Zhang, Kaiyang Zhou, and Ziwei Liu. Neural prompt search. *arXiv preprint arXiv:2206.04673*, 2022. 1

[19] Kaiyang Zhou, Jingkang Yang, Chen Change Loy, and Ziwei Liu. Conditional prompt learning for vision-language models. In *CVPR*, pages 16816–16825, 2022. 1