Resource-efficient Inference with Foundation Model Programs

Lunyiu Nie [1] **Zhimin Ding** [2] **Kevin Yu** [2] **Marco Cheung** [1] **Christopher Jermaine** [2] **Swarat Chaudhuri** [1]

Abstract

The inference-time resource costs of large language and vision models present a growing challenge in production deployments. We propose the use of *foundation model programs*, i.e., programs that can invoke foundation models with varying resource costs and performance, as an approach to this problem. Specifically, we present a method that translates a task into a program, then learns a policy for resource allocation that, on each input, selects foundation model "backends" for each program module. The policy uses smaller, cheaper backends to handle simpler subtasks, while allowing more complex subtasks to leverage larger, more capable models. We evaluate the method on two new "streaming" visual question-answering tasks in which a system answers a question on a sequence of inputs, receiving ground-truth feedback after each answer. Compared to monolithic multi-modal models, our implementation achieves up to 98% resource savings with minimal accuracy loss, demonstrating its potential for scalable and resource-efficient multi-modal inference [1].

1. Introduction

Foundation models (FMs) have reshaped the landscape of machine learning over the past few years, demonstrating unprecedented capabilities in language understanding (Achiam et al., 2023; Dubey et al., 2024), complex reasoning (Lu et al., 2024; Gupta et al., 2024), and multi-modal tasks (Li et al., 2022a; Liu et al., 2024). While much of the community's attention has focused on their training costs, the *inference-time resource use* of FMs is increasingly becoming a practical bottleneck. For commercial applications that require real-time responses — for instance, continuous streams of user queries to a multi-modal large language model (MLLM) — computational overhead and high latency can severely degrade user experience and inflate operational expenses (Xu et al., 2024).

In this paper, we propose the use of *foundation model programs* (FMPs) — code in Python-like languages that can call into a variety of specialized vision and language models as subroutines — to address this problem. Such programs have been previously motivated on the basis of the interpretability and flexibility they bring to multi-step tasks (Surís et al., 2023; Gupta & Kembhavi, 2023; Subramanian et al., 2023). Our insight is that they can also enable fine-grained decisions about resource allocation: simpler subtasks can rely on smaller, cheaper backends while more complex components can leverage larger, more capable models.

Concretely, we propose a framework of *resource-efficient foundation model programming* in which a task is automatically translated into an FMP that captures subtask dependencies and conditional control flow. Each submodule of the program is then assigned to one of several backend models, differing in resource cost and capability. For example, in Figure 1, a visual question answering (VQA) system receives the query "*Is there a cat sitting or laying on a laptop keyboard?*" Here, our method generates a program that uses a small, inexpensive object detection model to check whether both a cat and a laptop are present. Only if that condition is met does it invoke a more powerful vision-language model (VLM) for finer-grained reasoning.

We specifically focus on "streaming" tasks in which the system repeatedly solves a task — for example, answering a question — on a *sequence* of inputs. Each answer is provided without prior knowledge of the ground truth, and the system receives ground-truth feedback after each answer. In such settings, the cost of using a monolithic model is proportional to the number of inputs processed. By contrast, our approach uses the feedback from the early answers to learn a *policy* that dynamically selects which backend model to invoke for each subtask, conditioned on the program input. Specifically, we use a combination of a structured REINFORCE estimator and gradient-based Thompson Sampling to learn this policy.

While existing routing or cascading strategies (Chen et al., 2023; Shnitzer et al., 2023; Lu et al., 2023; Nie et al., 2024) attempt to reduce large language model (LLM) inference

[1]The University of Texas at Austin [2]Rice University. Correspondence to: Lunyiu Nie <lynie@utexas.edu>, Swarat Chaudhuri <swarat@cs.utexas.edu>. Presented at the *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models at ICML 2025*, Vancouver, Canada. PMLR 267, 2025.

[1]Source code and benchmarks are available at https://github.com/Flitternie/FMProgramming.

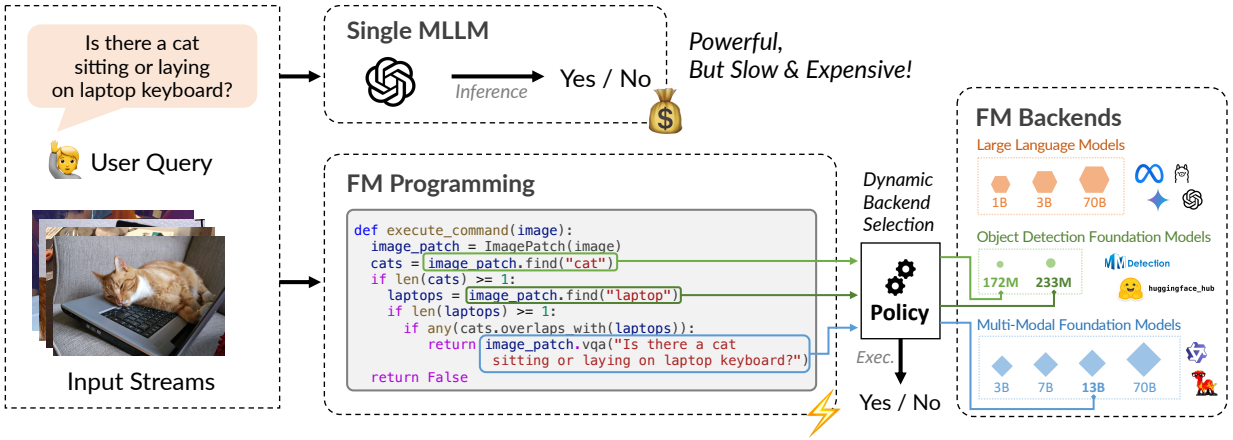

Figure 1: Illustration of a foundation model program synthesizing a VQA task by decomposing the task into sub-components. At runtime, the resource-efficient FM programming framework dynamically selects FM backends based on the task and input complexity to optimize accuracy and resource efficiency in real-time processing.

overheads by switching between model sizes, they do not exploit the rich structural dependencies that arise in complex, compositional workflows. By contrast, our programs make these dependencies explicit, opening up opportunities for more flexible resource optimization.

Given the lack of standard benchmarks for resource-efficient sequential decision-making, we evaluate our approach on two newly introduced benchmarks: (1) a streaming binary VQA benchmark, where the questions require yes/no answers, spanning 33 compositional reasoning tasks with over 2,000 annotated images per task; and (2) a streaming open-form VQA benchmark, involving diverse questions with a broader answer space, covering 50 tasks with 500 annotated images per task. Experimental results show that our FMP-based system consistently reduces inference costs by 50% to 98% compared to one-size-fits-all baselines, without compromising task accuracy.

In summary, our contributions are as follows:

- We propose the use of *foundation model programs* as a flexible approach to cost-efficient inference for complex, multi-modal workflows.
- We give a specific method for learning such programs in a sequential decision-making setting. The highlight of the method is an online resource allocation method that systematically trades off the resource consumption and performance of models in an input-dependent way.
- We release two streaming benchmarks for binary and open-form VQA, reflecting real-world tasks where inputs arrive sequentially at scale and resource-efficiency is key.
- We show empirical results on these benchmarks, which demonstrate that our program-based approach can achieve up to 98% cost savings with minimal accuracy degradation.

2. Problem Formulation

We consider *foundation model programs* (FMPs), which are neurosymbolic programs that interleave symbolic control flow with calls to a fixed set of generic neural functions $F = \{f_1, f_2, \ldots, f_K\}$. For instance, the program in Figure 1 uses functions `ImagePatch.find()` and `ImagePatch.vqa()`. Each f_k has multiple backend models $M_k = \{m_{k,1}, m_{k,2}, \ldots, m_{k,n_k}\}$ with varying accuracy and computational costs.

For a program with N calls to these functions, denoted by $\langle f_{k_1}, f_{k_2}, \ldots, f_{k_N} \rangle$, we define a *program configuration vector* $\vec{v} = \langle m_{k_1,j_1}, \ldots, m_{k_N,j_N} \rangle$ that specifies which backend to use for each call.

Given a sequence of input-output pairs $\{x_t, y_t\}_{t=1}^T$, our objective is to learn a policy π that maps each input x_t to a program configuration vector $\vec{v}_t = \pi(x_t)$ to maximize the cumulative reward $\sum_{t=1}^T R(\vec{v}_t, x_t, y_t)$, where

$$R(\vec{v}_t, x_t, y_t) = -\mathcal{L}(p(x_t|\vec{v}_t), y_t) - \lambda \mathcal{C}(\vec{v}_t).$$

Here, $\mathcal{L}$ measures prediction error between the program execution $p(x_t|\vec{v}_t)$ and reference y_t, $\mathcal{C}$ denotes the computational cost of invoked backends from $\vec{v}_t$, and $\lambda > 0$ balances accuracy-cost trade-offs. This optimization is performed *online*, with decisions made sequentially by policy π without prior knowledge of y_t. More details of the formulation are in Appendix A.1.

3. Foundation Model Programming

Our approach to addressing this problem consists of two key phases: *offline code generation* and *online resource allocation*. In the offline phase, we use a large language model (LLM) to synthesize a foundation model program (FMP) based on user specifications. This FMP comprises a se-

quence of generic neural function calls $\langle f_{k_1}, f_{k_2}, \ldots, f_{k_N} \rangle$, where each f_{k_i} can be executed by various backend models in M_{k_i} differing in accuracy and computational cost.

During the online phase, we dynamically assign a program configuration vector $\vec{v}_t$ to each input x_t, selecting a specific backend for each function call. Given the combinatorial explosion of possible configurations with N functions, we decompose the decision-making into N sub-policies π_{k_i}, each handling one function call. To facilitate this, we define a sub-reward function for each function call f_{k_i}:

$$r_{k_i,j_i} = -\lambda \mathcal{C}(m_{k_i,j_i}) - \frac{1}{N}\mathcal{L}(p(x_t|\vec{v}_t), y_t),$$

for all $m_{k_i,j_i} \in \vec{v}_t$. This sub-reward decomposes the global reward $R(\vec{v}_t, x_t, y_t)$ into structured contributions, including the local computational cost and a portion of the predictive loss. This decomposition reduces the optimization space and enables efficient backend selection.

Each sub-policy π_{k_i} predicts rewards for its associated backends, balancing exploration and exploitation via gradient-based Thompson Sampling (Zhang et al., 2020). For every backend m_{k_i,j_i}, the sub-policy samples a reward from a distribution reflecting both predicted reward and uncertainty, choosing the backend with the highest sampled value. This process constructs $\vec{v}_t$ incrementally across all function calls.

Due to the non-differentiable program structure, the sub-policies are trained using a structured REINFORCE algorithm (Williams, 1992) to estimate the policy gradient:

$$\nabla_{\theta_{k_i}} \mathcal{J}(\pi_{k_i}) \approx \sum_{t=1}^{T} \sum_{s=1}^{S} \nabla_{\theta_{k_i}} \log \pi_{k_i}(m^{(s)}_{k_i,j_i^*} | x_t^{(s)}; \theta_{k_i}) \cdot r^{(s)}_{k_i,j_i^*}.$$

This gradient updates the sub-policy parameters based on S sampled trajectories, leveraging the decomposed sub-rewards $r^{(s)}_{k_i,j_i^*}$ observable after program execution.

A detailed description and pseudocode of the full framework is deferred to Appendix A.2. This methodology provides a tractable solution to the online resource allocation challenge, with theoretical no-regret guarantees in Appendix B. By separating offline synthesis from online optimization and employing a decomposed policy structure, we achieve both flexibility and efficiency in program execution.

4. Benchmark

Motivated by the need for structured, sequential evaluation beyond the single-image-per-query setups typical of existing visual question answering (VQA) datasets (Goyal et al., 2017), we introduce two novel streaming VQA benchmarks.

Our first **Streaming Binary VQA** benchmark focuses on yes/no question answering, a task commonly studied in previous works (Antol et al., 2015; Zhang et al., 2016; Hudson & Manning, 2019). In this benchmark, systems are challenged to determine whether a sequence of images satisfies complex, compositional queries. These queries incorporate diverse reasoning types—spatial (*e.g.*, "Is there a person riding a bicycle next to a bus on the street?"), logical (*e.g.*, "Are there people riding bikes, scooters, or motorcycles while holding or using umbrellas?"), and numerical (*e.g.*, "Are there at least four horses on a beach?")—to better reflect real-world reasoning demands. The final benchmark includes 33 queries with more than 2000 annotated images for each query, featuring a realistic class imbalance setup. Further details on the benchmark construction and evaluation are provided in Appendix D.1.

Our second benchmark, **Streaming Open-form VQA**, evaluates a system's ability to answer open-form questions for a sequence of input images. This benchmark spans five reasoning categories: spatial (*e.g.*, "What is in the jar to the left of the juice?"), logical (*e.g.*, "What is the black object on the desk that is not electronic?"), numerical (*e.g.*, "How many extra bottles of beer do we need to make it a half dozen?"), comparative (*e.g.*, "Which bottle is taller, the left one or the right one?"), and external knowledge reasoning (*e.g.*, "How many states are there in the country whose flag is shown?"). Images are generated using a diffusion model with a dedicated pipeline to ensure diversity and quality control. To evaluate model robustness, we also introduce unanswerable images that are visually similar to the query but semantically invalid for answering. The final benchmark includes 50 queries with 500 annotated images per query. Complete details of the image generation pipeline, neurosymbolic program synthesis, and evaluation metrics (exact match accuracy) are described in Appendix D.2.

5. Experiments

5.1. Baselines

We compare our system against several baselines to establish its effectiveness: **(a) Single MLLMs.** We evaluate our system against the state-of-the-art multi-modal LLMs (MLLMs) that integrate both vision and language reasoning capabilities (Bai et al., 2025); **(b) MLLM Routing.** As an alternative adaptive strategy, a multi-armed bandit dynamically routes user queries to multiple MLLMs of varying sizes with different cost-accuracy trade-offs based on estimated rewards, balancing exploration and exploitation (Nguyen et al., 2024; Li, 2025). However, it does not account for the task structures in user queries; **(c) Static FM Program Configurations.** A common approach to resource management is to use a fixed, pre-determined configuration of foundation models for the FM programs without dynamic backend selection. Given the combinatorial space of configurations, we implement two variants: one using the cheapest FM configurations and another using the most

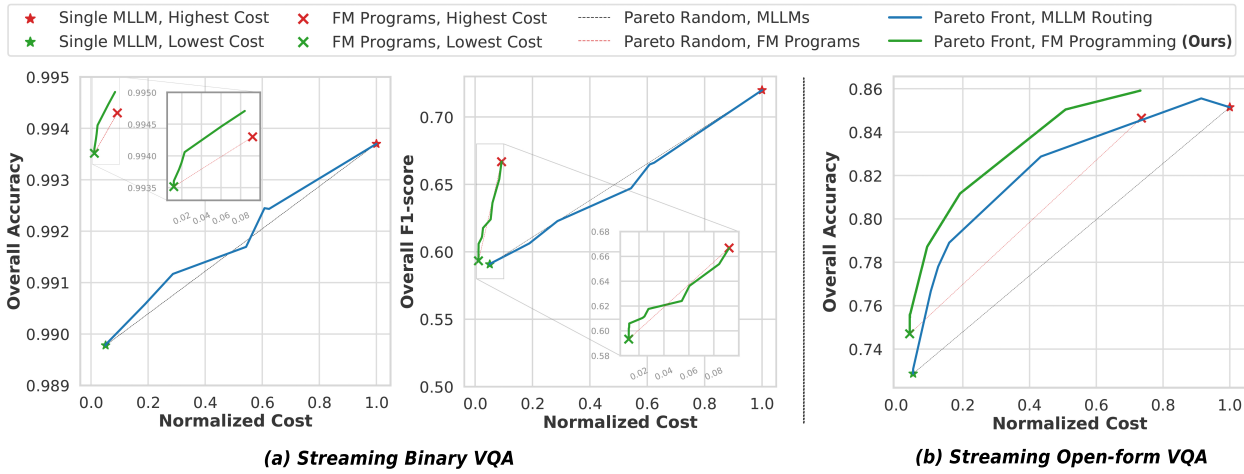

Figure 2: Experimental results on the (a) *Streaming Binary VQA* benchmark, and (b) *Streaming Open-form VQA* benchmark. Costs are normalized based on the inference costs of the most expensive MLLM, *i.e.*, Qwen2.5-VL 72B.

expensive configurations; **(d) Pareto-Random Routing.** Following the prior works (Hu et al., 2024; Jitkrittum et al., 2025), we employ a straightforward yet effective Pareto-random routing strategy through linear interpolation. We implement this approach separately for two scenarios: multimodal LLMs and static FM program configurations.

5.2. Implementation Details and Experimental Setups

In the experiments reported in the main paper, we consistently use Qwen2.5-VL (3B and 72B) for both the Single MLLM and MLLM Routing baselines. The FM program backends consist of Grounding-DINO Tiny (172M) and Base (224M) for object detection, along with Qwen2.5-VL 3B and 72B for vision-language understanding. We implement the structured policy using ResNet-18 with $\sim$11M parameters (He et al., 2016), ensuring the policy training and inference overheads do not exceed the resource savings. We implement an instrumentation system that analyzes the programs to determine function calls, modifies the abstract syntax tree to inject program configurations, and executes the modified program while collecting performance metrics including the execution traces and the computational costs.

5.3. Results

Streaming Binary VQA Results. As shown in Figure 2, the Pareto frontier of FM Programming (green line) significantly surpasses both the Pareto Random baseline (red dashed line) and the MLLM routing baseline (blue line). The FM Programming approach can achieve an accuracy comparable to the largest MLLM while reducing computational costs by over 98%. This striking efficiency gain stems from its ability to exploit task structure, where lightweight object detection modules filter out most negative samples, minimizing expensive MLLM inference. While its maximum F1 score is lower than MLLM-based methods due to conservative object detection thresholds prioritizing precision over recall to minimize costly false positives (Figure 5), this trade-off can be tuned via the reward function.

To evaluate robustness, we conduct additional experiments using an alternative set of FM backends with reduced capabilities. As shown in Figure 6, FM Programming maintains Pareto dominance, highlighting its robustness and adaptability across varied configurations.

Streaming Open-form VQA Results. As Figure 2 (b) shows, in the more challenging Streaming Open-form VQA benchmark, FM Programming also outperforms the Pareto Random and the MLLM routing baselines. Its dynamic backend allocation proves especially effective in low-cost regimes, where small cost increases lead to substantial accuracy gains. Compared to the largest MLLM, FM Programming achieves up to 50% cost savings without sacrificing performance—and in some cases, it even exceeds the accuracy while cutting costs by 30%. These results underscore FM Programming's ability to exploit task and input structure for optimal cost-efficiency, demonstrating its scalability and potential for real-world, resource-constrained applications.

6. Conclusion

We introduced the first framework using foundation model programs (FMPs) to dynamically optimize performance-resource trade-offs in multimodal reasoning tasks. Our experiments show substantial computational savings with minimal performance loss. Future work includes extending FMPs to broader agentic scenarios and advancing methods for joint learning of program structures and configurations.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *Journal of Machine Learning Research*, 22(98):1–76, 2021.

AgentLego, C. Agentlego: Open-source tool api library to extend and enhance llm agents, december 2023. *URL https://github. com/InternLM/agentlego*, 2023.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pp. 2425–2433, 2015.

Bai, S., Chen, K., Liu, X., Wang, J., Ge, W., Song, S., Dang, K., Wang, P., Wang, S., Tang, J., et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Ban, Y., He, J., and Cook, C. B. Multi-facet contextual bandits: A neural network perspective. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 35–45, 2021.

Black Forest Labs. Flux. https://github.com/black-forest-labs/flux, 2023.

Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J., et al. Mmdetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

Chen, L., Zaharia, M., and Zou, J. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., and Parikh, D. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6904–6913, 2017.

Gupta, N., Narasimhan, H., Jitkrittum, W., Rawat, A. S., Menon, A. K., and Kumar, S. Language model cascades: Token-level uncertainty and beyond. *arXiv preprint arXiv:2404.10136*, 2024.

Gupta, T. and Kembhavi, A. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14953–14962, 2023.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hu, Q. J., Bieker, J., Li, X., Jiang, N., Keigwin, B., Ranganath, G., Keutzer, K., and Upadhyay, S. K. Routerbench: A benchmark for multi-llm routing system. *arXiv preprint arXiv:2403.12031*, 2024.

Hudson, D. A. and Manning, C. D. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6700–6709, 2019.

Hurst, A., Lerer, A., Goucher, A. P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Jain, A., Kothyari, M., Kumar, V., Jyothi, P., Ramakrishnan, G., and Chakrabarti, S. Select, substitute, search: A new benchmark for knowledge-augmented visual question answering. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2491–2498, 2021.

Jitkrittum, W., Narasimhan, H., Rawat, A. S., Juneja, J., Wang, Z., Lee, C.-Y., Shenoy, P., Panigrahy, R., Menon, A. K., and Kumar, S. Universal model routing for efficient llm inference. *arXiv preprint arXiv:2502.08773*, 2025.

Johnson, J., Hariharan, B., Van Der Maaten, L., Fei-Fei, L., Lawrence Zitnick, C., and Girshick, R. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2901–2910, 2017.

Kreikemeyer, J. N. and Andelfinger, P. Smoothing methods for automatic differentiation across conditional branches. *IEEE Access*, 2023.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Li, J., Li, D., Xiong, C., and Hoi, S. Blip: Bootstrapping language-image pre-training for unified vision-language

understanding and generation. In *International conference on machine learning*, pp. 12888–12900. PMLR, 2022a.

Li, J., Li, D., Savarese, S., and Hoi, S. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pp. 19730–19742. PMLR, 2023.

Li, L. H., Zhang, P., Zhang, H., Yang, J., Li, C., Zhong, Y., Wang, L., Yuan, L., Zhang, L., Hwang, J.-N., et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10965–10975, 2022b.

Li, Y. Llm bandit: Cost-efficient llm generation via preference-conditioned dynamic routing. *arXiv preprint arXiv:2502.02743*, 2025.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755. Springer, 2014.

Liu, H., Li, C., Li, Y., and Lee, Y. J. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 26296–26306, 2024.

Lu, K., Yuan, H., Lin, R., Lin, J., Yuan, Z., Zhou, C., and Zhou, J. Routing to the expert: Efficient reward-guided ensemble of large language models. *arXiv preprint arXiv:2311.08692*, 2023.

Lu, P., Peng, B., Cheng, H., Galley, M., Chang, K.-W., Wu, Y. N., Zhu, S.-C., and Gao, J. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Nguyen, Q. H., Hoang, D. C., Decugis, J., Manchanda, S., Chawla, N. V., and Doan, K. D. Metallm: A high-performant and cost-efficient dynamic framework for wrapping llms. *arXiv preprint arXiv:2407.10834*, 2024.

Nie, L., Ding, Z., Hu, E., Jermaine, C., and Chaudhuri, S. Online cascade learning for efficient inference over streams. In *Forty-first International Conference on Machine Learning*, 2024.

OpenAI. Gpt-4o mini: Advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence, 2024.

Schwenk, D., Khandelwal, A., Clark, C., Marino, K., and Mottaghi, R. A-okvqa: A benchmark for visual question answering using world knowledge. In *European conference on computer vision*, pp. 146–162. Springer, 2022.

Shnitzer, T., Ou, A., Silva, M., Soule, K., Sun, Y., Solomon, J., Thompson, N., and Yurochkin, M. Large language model routing with benchmark datasets. *arXiv preprint arXiv:2309.15789*, 2023.

Subramanian, S., Narasimhan, M., Khangaonkar, K., Yang, K., Nagrani, A., Schmid, C., Zeng, A., Darrell, T., and Klein, D. Modular visual question answering via code generation. *arXiv preprint arXiv:2306.05392*, 2023.

Surís, D., Menon, S., and Vondrick, C. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11888–11898, 2023.

Wang, P., Yang, A., Men, R., Lin, J., Bai, S., Li, Z., Ma, J., Zhou, C., Zhou, J., and Yang, H. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International conference on machine learning*, pp. 23318–23340. PMLR, 2022.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

Xu, M., Yin, W., Cai, D., Yi, R., Xu, D., Wang, Q., Wu, B., Zhao, Y., Yang, C., Wang, S., et al. A survey of resource-efficient llm and multimodal foundation models. *arXiv preprint arXiv:2401.08092*, 2024.

Zhang, P., Goyal, Y., Summers-Stay, D., Batra, D., and Parikh, D. Yin and yang: Balancing and answering binary visual questions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5014–5022, 2016.

Zhang, W., Zhou, D., Li, L., and Gu, Q. Neural thompson sampling. *arXiv preprint arXiv:2010.00827*, 2020.

Zinkevich, M. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)*, pp. 928–936, 2003.

A. Supplementary Information

A.1. Problem Formulation

Foundation Model Programs. We consider programs written in a language such as Python, potentially synthesized

by an LLM. The programs are *neurosymbolic* because they interleave symbolic control flow with calls to a fixed set of *generic neural functions* $F = \{f_1, f_2, \ldots, f_K\}$, where each f_k denotes a high-level functionality (e.g., object detection, visual question answering, natural language understanding). For example, the program in Figure 1 makes calls to two generic neural functions `ImagePatch.find()` and `ImagePatch.vqa()`. In practice, we are interested in the case where the neural functions are implemented via foundation models. Hence, we use the term *foundation model program* (FMP) to refer to such programs.

Each generic function f_k in an FMP has an associated set of n_k *backend models* that can be used to implement it, namely $M_k = \{m_{k,1}, m_{k,2}, \ldots, m_{k,n_k}\}$, where each backend $m_{k,j}$ has different trade-offs between accuracy and computational cost. These backends may span a spectrum of models, from lightweight task-specific models to large, general-purpose language or multimodal models. Without loss of generality, we assume that the cost of invoking a backend is fixed and independent of the specific input it processes.

Now, assume that we analyze the program and produce an arbitrary ordering for the total of N calls to generic neural functions, such that k_i is the identity of the generic neural function associated with the i-th call in the program. Note that a program can call the same neural function multiple times with different inputs or arguments. The list of neural functions called by the program is then $\langle f_{k_1}, f_{k_2}, \ldots, f_{k_N} \rangle$. In our example program, the list is $\langle$`ImagePatch.find()`, `ImagePatch.find()`, `ImagePatch.vqa()`$\rangle$.

Further, assume the host programming language has a runtime in which we are able to dynamically assign each of the N calls to generic neural functions to a particular backend. Let j_i denote which of the backends is selected for the i-th generic neural function call, where $j_i \in \{1, ..., n_{k_i}\}$. That is, we select the j-th available neural backend m_{k_i,j_i} for the i-th generic neural function call.

Thus, we can customize the behavior of an FMP to optimize for accuracy and runtime cost on a specific program input, by choosing a specific list of backends $\vec{v} = \langle m_{k_1,j_1}, ..., m_{k_N,j_N} \rangle$. We call $\vec{v}$ a *program configuration vector*. On input x, we use $p(x|\vec{v})$ to denote the output of the program, given that program configuration vector $\vec{v}$ was chosen.

Task Objective. The idea of using FMPs for resource use optimization can be instantiated in a wide range of problems. In this paper, we focus on settings in which the goal is to solve a task—for example, answer a question—on a sequence of input-output pairs $\{x_t, y_t\}_{t=1}^T$. We assume that the structure of the programs we use only depends on the overall task and not the specific inputs. Therefore, on the input x_t for time step t, we only need to decide on a suitable program configuration vector $\vec{v}_t$.

We want $\vec{v}_t$ to be such that the program execution output $p(x_t \mid \vec{v}_t)$ approximates the ground truth y_t, while minimizing execution cost. To capture this trade-off, we define the following reward function:

$$R(\vec{v}_t, x_t, y_t) = -\mathcal{L}(p(x_t|\vec{v}_t), y_t) - \lambda \mathcal{C}(\vec{v}_t),$$

where $\mathcal{L}$ quantifies the output discrepancy between $p(x_t \mid \vec{v}_t)$ and the ground truth y_t, $\mathcal{C}$ represents the actual computational cost incurred when running the program with configuration $\vec{v}_t$ on input x_t, and $\lambda > 0$ is a trade-off weighting factor. Importantly, due to control flow in the program, not all neural backends specified in $\vec{v}_t$ may be invoked on a given input; $\mathcal{C}$ accounts only for the cost of the operations actually executed.

Over T time steps, the objective is to learn a *policy* π that maps each input x_t to a program configuration vector $\vec{v}_t$. Let Π be the space of such policies. We seek to solve the problem

$$\max_{\pi \in \Pi} \sum_{t=1}^{T} R(\vec{v}_t, x_t, y_t) \quad \text{subject to} \quad \vec{v}_t = \pi(x_t). \tag{1}$$

Importantly, we require this optimization problem to be solved *online*. That is, we assume that our inputs arrive sequentially and require decisions to be made without prior knowledge of the ground truth. Only after the selected configuration is executed and the output $p(x_t \mid \vec{v}_t)$ is produced is the ground truth y_t revealed and the reward $R(\vec{v}_t, x_t, y_t)$ computed.

A.2. Methodology

Offline Code Generation. We begin by synthesizing a foundation model program p from the user specification using an LLM. This process produces a task-specific program sketch, including a sequence of generic neural function calls $\langle f_{k_1}, f_{k_2}, \ldots, f_{k_N} \rangle$, which defines the high-level structure of the computation, while the backend selection for these neural functions is determined online.

Online Resource Allocation. After offline synthesis, the main challenge is to select a program configuration vector $\vec{v}_t$ for each input x_t, dynamically assigning a neural backend $m_{k_i,j_i} \in M_{k_i}$ to each function call f_{k_i}. The space of possible configurations grows combinatorially with N, making exhaustive search intractable. Therefore, we propose a structured policy that decomposes this decision process into N manageable sub-policies, one per function call.

Specifically, for each function call f_{k_i}, we define a *sub-reward* function:

$$r_{k_i,j_i} = -\lambda \mathcal{C}(m_{k_i,j_i}) - \frac{1}{N} \mathcal{L}(p(x_t|\vec{v}_t), y_t),$$

Algorithm 1 Structured REINFORCE Framework

Initialize: Policy parameters θ_{k_i}, uncertainty estimates $\mathbf{U}_{k_i}$, learning rate η, exploration factor ν
for each input x_t **do**
  for each function call f_{k_i} in program p **do**
    Predict reward $r'_{k_i,j_i} = \pi_{k_i}(m_{k_i,j_i}|x_t;\theta_{k_i})$ for each FM backend m_{k_i,j_i}
    Compute uncertainty $\sigma_{k_i,j_i} = \sqrt{\sum_l \frac{g^2_{k_i,j_i,l}}{\mathbf{U}_{k_i,l}}}$
    Sample adjusted reward for exploration: $\hat{r}_{k_i,j_i} \sim \mathcal{N}\big(r'_{k_i,j_i}, (\nu \cdot \sigma_{k_i,j_i})^2\big)$
    Select FM backend m_{k_i,j_i^*} with highest sampled reward $\hat{r}_{k_i,j_i}$
    Update parameter uncertainties $\mathbf{U}_{k_i,l}$
  end for
  Execute program p with selected configuration $\vec{v}_t = (m_{k_1,j_1^*}, ..., m_{k_N,j_N^*})$
  Observe final reward $R(\vec{v}_t, x_t, y_t)$
  for each function call f_{k_i} **do**
    Compute policy gradient $\nabla_{\theta_{k_i}} \mathcal{J}(\pi_{k_i})$ based on observed reward
    Update policy parameters: $\theta_{k_i} \leftarrow \theta_{k_i} - \eta \nabla_{\theta_{k_i}} \mathcal{J}(\pi_{k_i})$
  end for
end for

for all $m_{k_i,j_i} \in \vec{v}_t$. This sub-reward decomposes the global reward $R(\vec{v}_t, x_t, y_t)$ into local contributions. It integrates the local computational cost $\mathcal{C}(m_{k_i,j_i})$ associated with backend m_{k_i,j_i}, and a portion of the predictive loss $\mathcal{L}(p(x_t|\vec{v}_t), y_t)$, that is determined once the entire configuration $\vec{v}_t$ is set.

To model these rewards, we define a *subpolicy* π_{k_i} with learnable parameters θ_{k_i}. Given the input x_t, subpolicy π_{k_i} outputs a reward prediction

$$r'_{k_i,j_i} = \pi_{k_i}(m_{k_i,j_i}|x_t;\theta_{k_i}) \quad \text{for each } m_{k_i,j_i} \in M_{k_i}.$$

This structured design simplifies the decision space: rather than jointly optimizing over all $n_{k_1} \times \cdots \times n_{k_N}$ backend combinations, we train N separate subpolicies. Each subpolicy is specialized to one of the N function calls, thereby simplifying the optimization process, enabling parallel learning, and reducing unwanted interference across calls.

Gradient-based Thompson Sampling. To balance exploration and exploitation in the online setting, decisions are made using Thompson Sampling (Zhang et al., 2020) instead of greedily selecting the FM backend with the highest predicted reward. For each backend $m_{k_i,j_i} \in M_{k_i}$, the subpolicy samples a reward from a normal distribution:

$$\hat{r}_{k_i,j_i} \sim \mathcal{N}\big(r'_{k_i,j_i}, (\nu \cdot \sigma_{k_i,j_i})^2\big),$$

where $\sigma_{k_i,j_i} = \sqrt{\sum_l \frac{g^2_{k_i,j_i,l}}{\mathbf{U}_{k_i,l}}}$ quantifies uncertainty in the reward prediction. Here, l indexes each individual parameter of the subpolicy, $g_{k_i,j_i,l}$ is the gradient of the reward prediction with respect to parameter l, $\mathbf{U}_{k_i}$ tracks the accumulated gradient-based parameter uncertainties, and ν scales the exploration. We select the backend with the highest sampled reward:

$$j_i^* = \underset{j_i \in \{1,\ldots,n_{k_i}\}}{\arg\max} \hat{r}_{k_i,j_i}.$$

After selection, the uncertainty parameter $\mathbf{U}_{k_i}$ is updated to refine future exploration:

$$\mathbf{U}_{k_i,l} \leftarrow \mathbf{U}_{k_i,l} + g^2_{k_i,j_i^*,l},$$

where $g_{k_i,j_i^*,l}$ is the gradient of the selected backend m_{k_i,j_i^*} with respect to parameter l.

We repeat the above process for each $i = 1, \ldots, N$, allowing each subpolicy to choose one backend per function call, thereby yielding the program configuration vector at time step t:

$$\vec{v}_t = \big(m_{k_1,j_1^*}, m_{k_2,j_2^*}, \ldots, m_{k_N,j_N^*}\big).$$

Structured REINFORCE Algorithm. We now describe the online learning of the subpolicies using only the global reward $R(\vec{v}_t, x_t, y_t)$ observed after execution. The learning objective for the overall policy $\pi = \{\pi_{k_1}, \ldots, \pi_{k_N}\}$ is to maximize cumulative reward over T episodes:

$$\mathcal{J}(\pi) = \sum_{t=1}^{T} R\big(\vec{v}_t, x_t, y_t\big), \quad \text{where } \vec{v}_t = \big(m_{k_1,j_1^*}, m_{k_2,j_2^*}, \ldots, m_{k_N,j_N^*}\big).$$

Since the reward $R(\vec{v}_t, x_t, y_t)$ is equivalent to the aggregation of all sub-rewards:

$$R(\vec{v}_t, x_t, y_t) = -\mathcal{L}\big(p(x_t|\vec{v}_t),\, y_t\big) - \lambda \mathcal{C}\big(\vec{v}_t\big)$$
$$= \sum_{i=1}^{N} \left[-\frac{1}{N} \mathcal{L}(p(x_t|\vec{v}_t), y_t) - \lambda \mathcal{C}(m_{k_i,j_i}) \right] = \sum_{i=1}^{N} r_{k_i,j_i^*},$$

we convert the learning objective into optimizing each subpolicy π_{k_i} independently:

$$\mathcal{J}\big(\pi_{k_i}\big) = \sum_{t=1}^{T} \mathbb{E}_{j_i^* \sim \pi_{k_i}(\cdot|x_t)} \left[r_{k_i,j_i^*} \right].$$

Because the program execution is non-differentiable due to control flow structures like conditional branches and loops (Kreikemeyer & Andelfinger, 2023), we employ the REINFORCE algorithm (Williams, 1992) to estimate policy gradients:

$$\nabla_{\theta_{k_i}} \mathcal{J}\big(\pi_{k_i}\big)$$
$$= \sum_{t=1}^{T} \mathbb{E}_{j_i^* \sim \pi_{k_i}(\cdot|x_t)} \left[\nabla_{\theta_{k_i}} \log \pi_{k_i}\big(m_{k_i,j_i^*} \mid x_t;\, \theta_{k_i}\big) \cdot r_{k_i,j_i^*} \right].$$

Note that each r_{k_i,j_i^*} depends on the full program execution but reflects a partial credit assignment for subpolicy π_{k_i}.

In practice, we approximate the expectation using S sampled trajectories:

$$\nabla_{\theta_{k_i}} \mathcal{J}(\pi_{k_i}) \approx \sum_{t=1}^{T} \sum_{s=1}^{S} \nabla_{\theta_{k_i}} \log \pi_{k_i}(m_{k_i,j_i^*}^{(s)} \mid x_t^{(s)}; \theta_{k_i}) \cdot r_{k_i,j_i^*}^{(s)}.$$

The sub-policies are periodically trained to stabilize learning:

$$\theta_{k_i} \leftarrow \theta_{k_i} - \eta \nabla_{\theta_{k_i}} \mathcal{J}(\pi_{k_i}),$$

where η is the learning rate. The overall framework is detailed in Algorithm 1.

B. No-Regret Guarantee for Structured REINFORCE

We establish that our proposed structured REINFORCE algorithm achieves a no-regret guarantee in an online learning setting. Before presenting the main theorem, we outline the key assumptions that underpin our analysis:

Assumption 1 (Bounded Rewards)**.** *For all time steps t, configurations $\vec{v}_t \in V$, and inputs $x_t \in \mathcal{D}$, the reward satisfies $R(\vec{v}_t, x_t, y_t) \in [R_{\min}, R_{\max}]$, where $R_{\max} - R_{\min} < \infty$.*

Assumption 2 (Policy Expressiveness)**.** *For each input x_t, there exists an optimal configuration $\vec{v}_{x_t}^* \in V$ that maximizes $R(\vec{v}, x_t, y_t)$. Moreover, the policy class is sufficiently expressive such that there exist parameters $\{\theta_{k_i}^*\}_{i=1}^N$ for which $\pi_{k_i}(m_{k_i,j_i}^* \mid x_t; \theta_{k_i}^*) \approx 1$, where m_{k_i,j_i}^* is the optimal backend for f_{k_i} in $\vec{v}_{x_t}^*$.*

Assumption 3 (Sufficient Exploration)**.** *The algorithm employs Thompson Sampling with an exploration parameter $\nu > 0$, ensuring that every backend $m_{k_i,j_i} \in M_{k_i}$ has a non-zero probability of being sampled at each time step t.*

Assumption 4 (Convergence of Policy Gradient)**.** *The learning rate η_t is set to $1/\sqrt{t}$, and the policy parameterization (e.g., softmax over M_{k_i}) guarantees that gradient updates converge to a near-optimal policy (Agarwal et al., 2021).*

Assumption 5 (Stationary Input Distribution)**.** *Inputs x_t are drawn independently and identically distributed (i.i.d.) from a fixed distribution $\mathcal{D}$, ensuring a consistent optimal policy over time.*

With these assumptions in place, we can formally state the main result:

Theorem 1. *Under Assumptions 1–5, the structured REINFORCE algorithm is no-regret, meaning that the average regret satisfies:*

$$\frac{\gamma_T}{T} \to 0 \quad as \quad T \to \infty,$$

both in expectation and with high probability, where the regret γ_T is defined as:

$$\gamma_T = \max_{\vec{v} \in V} \sum_{t=1}^{T} R(\vec{v}, x_t, y_t) - \sum_{t=1}^{T} R(\vec{v}_t, x_t, y_t).$$

We now prove this theorem, showing that the algorithm's regret diminishes over time. The proof proceeds by defining the regret, analyzing the algorithm's behavior under the assumptions, and bounding the regret both in expectation and with high probability.

Proof of Theorem 1

We start with giving the formal definition of regret γ_T, which measures the cumulative difference between the maximum achievable reward and the algorithm's actual reward over T steps:

$$\gamma_T = \max_{\vec{v} \in V} \sum_{t=1}^{T} R(\vec{v}, x_t, y_t) - \sum_{t=1}^{T} R(\vec{v}_t, x_t, y_t).$$

Our goal is to show that the average regret, $\frac{\gamma_T}{T}$, approaches zero as $T \to \infty$. To simplify the analysis, we use a stronger benchmark: the optimal context-dependent configuration $\vec{v}_{x_t}^*$ that maximizes $R(\vec{v}, x_t, y_t)$ for each x_t. Since $\max_{\vec{v} \in V} R(\vec{v}, x_t, y_t) \leq R(\vec{v}_{x_t}^*, x_t, y_t)$, we have:

$$\gamma_T \leq \sum_{t=1}^{T} R(\vec{v}_{x_t}^*, x_t, y_t) - \sum_{t=1}^{T} R(\vec{v}_t, x_t, y_t).$$

This upper bound focuses the proof on the gap between the optimal and achieved rewards per step.

The structured REINFORCE algorithm updates sub-policy parameters θ_{k_i} using policy gradients. The expected cumulative reward is:

$$\mathcal{J}(\pi) = \mathbb{E}_\pi \left[\sum_{t=1}^{T} R(\vec{v}_t, x_t, y_t) \right],$$

with the gradient for each sub-policy:

$$\nabla_{\theta_{k_i}} \mathcal{J}(\pi_{k_i}) = \sum_{t=1}^{T} \mathbb{E}_{j_i^* \sim \pi_{k_i}(\cdot \mid x_t)} \left[\nabla_{\theta_{k_i}} \log \pi_{k_i}(m_{k_i,j_i^*} \mid x_t; \theta_{k_i}) \cdot r_{k_i,j_i^*} \right].$$

The algorithm approximates this gradient with a single sample, updating parameters as:

$$\theta_{k_i,t+1} = \theta_{k_i,t} + \eta_t \nabla_{\theta_{k_i}} \log \pi_{k_i}(m_{k_i,j_i^*} \mid x_t; \theta_{k_i,t}) \cdot r_{k_i,j_i^*},$$

where $\eta_t = 1/\sqrt{t}$ (Assumption 4), and the sub-reward is defined as:

$$r_{k_i,j_i^*} = -\lambda \mathcal{C}(m_{k_i,j_i^*}) - \frac{1}{N} \mathcal{L}(p(x_t \mid \vec{v}_t), y_t).$$

Assumptions 2 (expressive policy class) and 3 (sufficient exploration via Thompson Sampling) ensure that each sub-policy π_{k_i} converges to the optimal sub-policy $\pi^*_{k_i}$, where $\pi^*_{k_i}(m^*_{k_i,j_i} \mid x_t) \approx 1$ for the optimal backend $m^*_{k_i,j_i}$ in $\vec{v}^*_{x_t}$. Thus, the overall policy π_t converges to the optimal policy π^*, leading to $\mathbb{E}_{\pi_t}[R(\vec{v}_t, x_t, y_t)] \to R(\vec{v}^*_{x_t}, x_t, y_t)$ as $t \to \infty$.

Define the per-step regret as:

$$\rho_t = R(\vec{v}^*_{x_t}, x_t, y_t) - R(\vec{v}_t, x_t, y_t),$$

with expected value:

$$\mathbb{E}[\rho_t] = \mathbb{E}_{x_t \sim \mathcal{D}}\left[R(\vec{v}^*_{x_t}, x_t, y_t) - \mathbb{E}_{\vec{v}_t \sim \pi_t}[R(\vec{v}_t, x_t, y_t)]\right].$$

Since $\pi_t \to \pi^*$, we have $\mathbb{E}[\rho_t] \to 0$. The total expected regret is:

$$\mathbb{E}[\gamma_T] \leq \sum_{t=1}^{T} \mathbb{E}[\rho_t].$$

Given $\eta_t = 1/\sqrt{t}$ and standard policy gradient convergence (Assumption 4), we bound:

$$\mathbb{E}[\gamma_T] \leq C\sqrt{T},$$

for some constant C based on the reward bounds and policy parameters (Zinkevich, 2003; Ban et al., 2021). Thus, the average expected regret satisfies:

$$\frac{\mathbb{E}[\gamma_T]}{T} \leq \frac{C}{\sqrt{T}} \to 0 \quad \text{as} \quad T \to \infty,$$

establishing no-regret in expectation.

Now, we extend this to show that the actual regret γ_T converges similarly with high probability. By Assumption 1, rewards are bounded, so $\rho_t \in [-(R_{\max} - R_{\min}), R_{\max} - R_{\min}]$, and let $B = R_{\max} - R_{\min}$. Since x_t are i.i.d. (Assumption 5) and $\vec{v}_t$ are sampled independently given x_t and π_t, the ρ_t are independent. Applying Hoeffding's inequality to $\gamma_T = \sum_{t=1}^{T} \rho_t$:

$$\mathbb{P}\left(|\gamma_T - \mathbb{E}[\gamma_T]| \geq \epsilon\right) \leq 2\exp\left(-\frac{2\epsilon^2}{TB^2}\right).$$

Set $\epsilon = \delta T$, so:

$$\mathbb{P}\left(|\gamma_T - \mathbb{E}[\gamma_T]| \geq \delta T\right) \leq 2\exp\left(-\frac{2\delta^2 T}{B^2}\right).$$

This probability approaches 0 exponentially as $T \to \infty$. Thus, with probability at least $1 - 2\exp\left(-\frac{2\delta^2 T}{B^2}\right)$, which approaches 1 as T grows, we have:

$$\gamma_T < \mathbb{E}[\gamma_T] + \delta T \leq C\sqrt{T} + \delta T.$$

Dividing by T, we find:

$$\frac{\gamma_T}{T} < \frac{C}{\sqrt{T}} + \delta.$$

For any $\epsilon > 0$, choose $\delta = \frac{\epsilon}{2}$ and $T > \left(\frac{2C}{\epsilon}\right)^2$, so $\frac{C}{\sqrt{T}} < \frac{\epsilon}{2}$, yielding:

$$\frac{\gamma_T}{T} < \epsilon,$$

with probability approaching 1. Hence, $\frac{\gamma_T}{T} \to 0$ with high probability, completing the no-regret proof.

C. Detailed Experimental Setups

All experiments are conducted on a single machine equipped with 8 NVIDIA A40 GPUs (48GB memory each), running CUDA 12.4.

The current FM backend is set up with VLLM (Kwon et al., 2023), Huggingface Hub, MMDetection (Chen et al., 2019), and AgentLego (AgentLego, 2023). Our framework supports a flexible backend construction with any open-source or closed-source API-based models. However, due to the unavailability of computational costs of closed-source models, such as GPT and Gemini series models, we do not include them in our FM backend during experiments.

Detailed package versions are listed in the environment file available at `https://github.com/Flitternie/FMProgramming/environment.yml`. Hyperparameter settings are specified in the experimental configuration files for `binary VQA` and `open-form VQA`.

For open-form VQA experiments, we consistently use the same system prompt for MLLMs:

System Prompt 1

```
Keep your answer short.  Try to
answer within 3 words.  For numerical
answers, use number digits (e.g., 5
instead of five), and returns the
number only.  If there are multiple
answers, separate them with a comma
(e.g., cat, dog).  If you find the
question unanswerable based on the
image content, output "N/A".  For
example, if the image content is
irrelevant to the question, or the
content in the image does not fully
and clearly match all the entities,
humans, attributes, spatial, logical,
and numerical constraints in the
question, output "N/A".
```

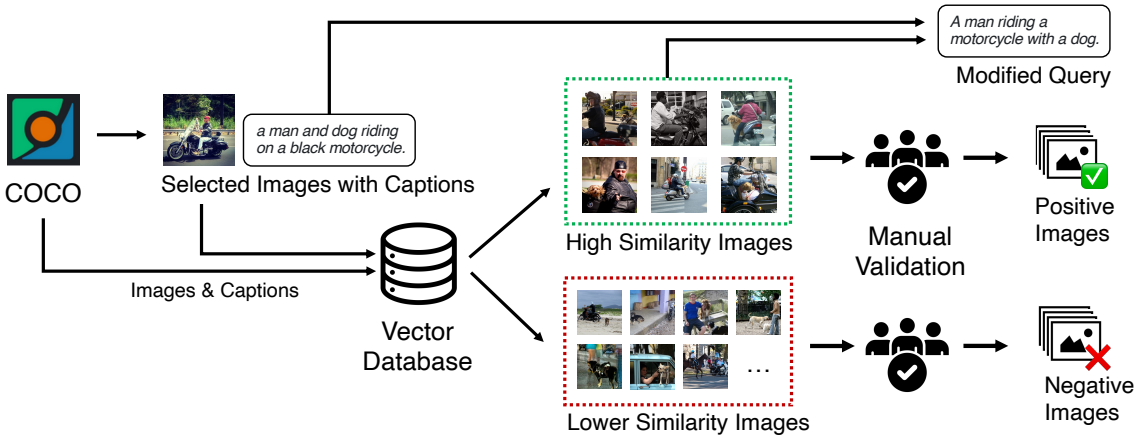

Figure 3: Benchmark construction pipeline for the Streaming Binary VQA dataset.

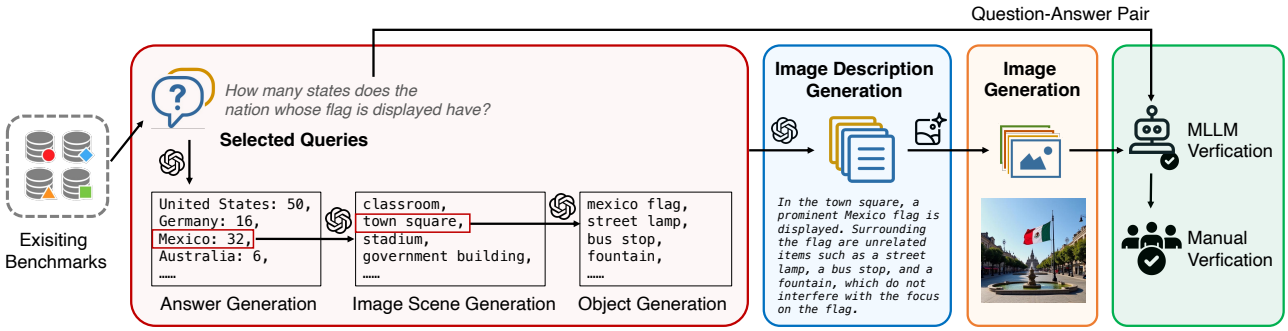

Figure 4: Benchmark construction pipeline for the Streaming Open-form VQA dataset.

D. Detailed Benchmark Construction and Evaluation

D.1. Streaming Binary VQA

Benchmark Construction. Our dataset is constructed from COCO (Lin et al., 2014), selecting captions that require multi-object compositional reasoning with spatial, logical, or numerical constraints. For each query, we prepare a set of more than 2,000 images, sampled based on the similarity of their captions to the query. The system must output a binary decision (*yes/no*) for each image indicating whether it satisfies the compositional query. To enforce structured reasoning, we leverage an LLM[2] to generate FM programs in a predefined DSL. These programs decompose the query into discrete reasoning steps, guiding the selection of foundation models for subtask execution. A detailed pipeline for benchmark construction is illustrated in Figure 3.

Annotation and Verification. Eleven human annotators validate the correctness of the neurosymbolic programs and the image labels, ensuring that each program aligns with the intended reasoning process and each image is correctly labeled against the compositional constraints. The final dataset consists of 33 queries covering three primary reasoning types (note that a query may fall into multiple types):

- **Spatial Reasoning** (20 queries): These queries require understanding and interpreting the spatial relationships between objects or people in an image. They often describe where things are located relative to each other, *e.g.*, "*Is there a person riding a bicycle next to a bus on the street*".
- **Logical Reasoning** (15 queries): These queries involve conditions, attributes, or combinations that require deductive thinking. The model needs to process logical relationships, such as inclusion, exclusion, or conjunction, *e.g.*, "*Are there people riding bikes, scooters, or motorcycles while holding or using umbrellas?*".
- **Numerical Reasoning** (9 queries): These queries test the ability to understand and count quantities or numbers in a scene. They often specify exact counts or comparisons,

[2]We use `GPT-4o` (Hurst et al., 2024) as the LLM throughout this section unless otherwise specified.

		VQA v2.0	GQA	CLEVR	A-OKVQA	Streaming VQA	
						Binary	Open-form
Query	Multiple Objects	✗	✓	✓	✓	✓	✓
	Spatial Reasoning	✓	✓	✓	✓	✓	✓
	Logical Reasoning	✓	✓	✓	✓	✓	✓
	Numerical Reasoning	✓	✓	✓	✓	✓	✓
	Comparative Reasoning	✓	✓	✓	✓	✗	✓
	External Knowledge	✗	✗	✗	✓	✗	✓
Image	Source	COCO	COCO & Flickr	Synthetic	COCO	COCO	Generation
	Unanswerable Images	✗	✗	✗	✗	✗	✓
Scale	# Queries	1.1M	22M	999,968	24,903	33	50
	# Images	200K	113K	100,000	23,692	66,279	25,000
	# Image(s) per query	2	1	1	1	>**2000**	**500**

Table 1: Comparing to the existing VQA benchmarks (Goyal et al., 2017; Hudson & Manning, 2019; Johnson et al., 2017; Schwenk et al., 2022), Streaming VQA is the first that provides a sequence of images for each query.

e.g., "*Are there at least four horses on a beach*".

Evaluation. Since the benchmark is highly imbalanced, with a positive-to-negative ratio of around 1:100, task performance is measured using accuracy, recall, precision, and F1-score.

D.2. Streaming Open-form VQA

Benchmark Construction. To enable evaluation on more complex open-form questions, we construct a dataset comprising 50 queries and 25,000 images, spanning five distinct reasoning categories. To ensure the validity and diversity of these queries, we randomly sample them from established benchmark datasets. Specifically, spatial queries are drawn from the GQA dataset (Hudson & Manning, 2019), focusing on queries labeled as `relS` and `categoryRelS`. Logical queries are also sampled from GQA, targeting the detailed types `twoCommon`, `twoSameMaterialC`, `twoDifferentC`, and `twoDifferent`. Numerical queries are selected from the A-OKVQA dataset (Schwenk et al., 2022), while comparative and external knowledge queries are sourced from OKVQAS3 (Jain et al., 2021). To enhance clarity and naturalness, some of the queries are manually rewritten.

Each query is associated with 500 generated images. The image generation pipeline begins with an LLM generating a set of 10 possible answers for each query, proposing potential scene setups along with 3 additional objects, and constructing detailed image descriptions. These descriptions are then used to prompt a diffusion model[3] for image generation.

[3]We use `FLUX.1-dev` (Black Forest Labs, 2023) as the diffusion model for image generation.

To assess model robustness and reasoning precision, we incorporate unanswerable images that are visually coherent but semantically invalid with respect to the query. These include both unrelated (random) images and images that are intentionally crafted to closely resemble answerable cases, making them more difficult to distinguish. This setup challenges models not only to infer the correct answer when possible but also to recognize when a question cannot be answered from the image.

Additionally, for each query, we also synthesize the corresponding FM program in the predefined DSL, providing a structured decomposition of the reasoning process.

Annotation and Validation. A two-step validation process is employed. First, a multi-modal LLM, `GPT-4o-mini` (OpenAI, 2024), verifies each image by generating an answer and comparing it to the expected answer. Only images where the MLLM's response matches the assigned answer are retained. Then, human evaluators verify a random image subset, achieving approximately 93% accuracy. The final benchmark encompasses five reasoning types:

- **Spatial Reasoning** (13 queries): These questions require understanding the spatial relationships between objects within a scene, *e.g.*, "*What is in the jar to the left of the juice?*".
- **Logical Reasoning** (9 queries): This category involves applying conditions, rules, or filters to identify specific objects or answer complex queries, *e.g.*, "*What is the black object on the desk that is not electronic?*", "*How many people are wearing both glasses and a hat?*".
- **Numerical Reasoning** (11 queries): This category requires counting, comparing numbers, or calculating quantities based on visual information, *e.g.*, "*How many extra*

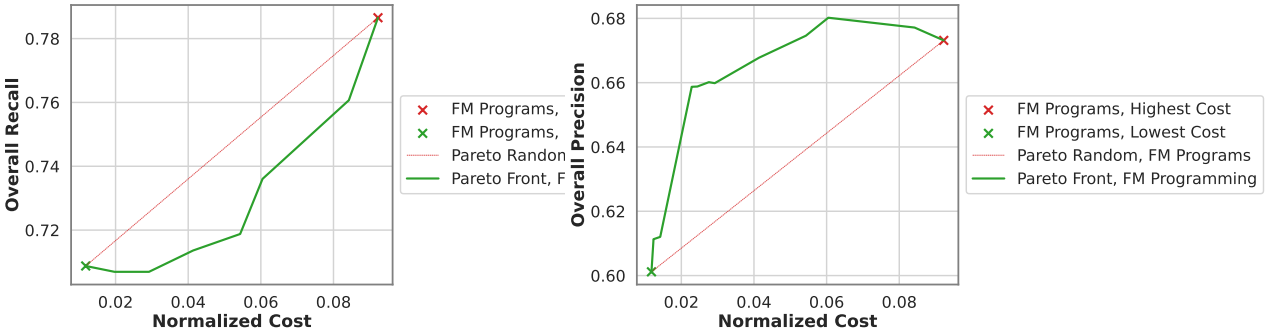

Figure 5: Pareto front of precision and recall scores on the Streaming Binary VQA benchmark. The policy prioritizes precision over recall as the cost budget increases.

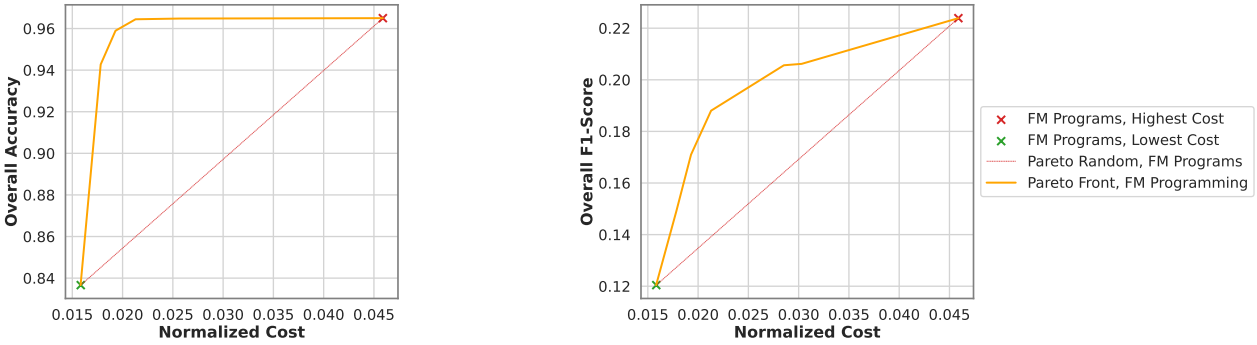

Figure 6: Additional results on the Streaming Binary VQA benchmark with different FM backends. Costs are normalized based on the inference costs of Qwen2.5-VL 72B. The backends include GLIP tiny (231M) and base (430M) models (Li et al., 2022b) for object detection, OFA base (182M) (Wang et al., 2022) and BLIP-2 OPT-2.7B (3.745B) (Li et al., 2023) for language-vision understanding. FM programming outperforms the baseline with better cost-performance trade-offs, demonstrating its adaptability across backend setups.

bottles of beer do we need to make it a half dozen?".

- **Comparative Reasoning** (11 queries): These questions involve evaluating two or more objects in terms of their attributes, such as size, height, quantity, or quality, *e.g.*, "*Which bottle is taller, the left one or the right one?*".
- **External Knowledge Reasoning** (6 queries): These questions rely on information that extends beyond what is immediately visible in the image, often drawing on common sense or factual knowledge, *e.g.*, "*The fruit in the picture is a good source of what vitamin?*", "*How many states are there in the country whose flag is being displayed?*".

Evaluation. Performance for the streaming VQA task is evaluated using exact match accuracy, measuring the proportion of questions answered correctly without partial credit.