2025 32nd Asia-Pacific Software Engineering Conference (APSEC) | 979-8-3315-6653-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/APSEC66846.2025.00061

Chart2Code-MoLA: Efficient Multi-Modal Code Generation via Adaptive Expert Routing

Yifei Wang, Jacky Keung, Zhenyu Mao, Jingyu Zhang*, Yuchen Cao
Department of Computer Science, City University of Hong Kong, Hong Kong, China
ywang4748-c@my.cityu.edu.hk, Jacky.Keung@cityu.edu.hk, zhenyumao2-c@my.cityu.edu.hk,
jzhang2297-c@my.cityu.edu.hk, cynthcao2-c@my.cityu.edu.hk

***Abstract*—Chart-to-code generation is a critical task in automated data visualization, translating complex chart structures into executable programs. While recent Multi-modal Large Language Models (MLLMs) improve chart representation, existing approaches still struggle to achieve cross-type generalization, memory efficiency, and modular design. To address these challenges, this paper proposes C2C-MoLA, a multimodal framework that synergizes Mixture of Experts (MoE) with Low-Rank Adaptation (LoRA). The MoE component uses a complexity-aware routing mechanism with domain-specialized experts and load-balanced sparse gating, dynamically allocating inputs based on learnable structural metrics like element count and chart complexity. LoRA enables parameter-efficient updates for resource-conscious tuning, further supported by a tailored training strategy that aligns routing stability with semantic accuracy. Experiments on Chart2Code-160k show that the proposed model improves generation accuracy by up to 17%, reduces peak GPU memory by 18%, and accelerates convergence by 20%, when compared to standard fine-tuning and LoRA-only baselines, particularly on complex charts. Ablation studies validate optimal designs, such as 8 experts and rank-8 LoRA, and confirm scalability for real-world multimodal code generation.**

***Index Terms*—Chart-to-Code Generation, Multi-Modal Learning, Mixture of Experts (MoE), Low-Rank Adaptation (LoRA)**

I. Introduction

With the growing reliance on data visualization for insight communication, automated understanding of visual content has become central to intelligent systems [1]. Chart-to-code generation refers to the task of converting visualizations such as bar, line, and scatter plots into executable programs, which requires precise interpretation of visual semantics and syntactically correct code generation [2]. Early methods used rule-based extraction from synthetic images or well-structured images [3] [4], leveraging predefined heuristics to identify chart elements. Subsequently, neural approaches introduced encoder-decoder architectures to tackle chart-to-data tasks [5] [6], learning to map pixel-level inputs to structured representations. These advances laid the foundation for multimodal code generation by bridging visual and symbolic modalities [7].

Recent progress has been driven by Multimodal Large Language Models (MLLMs) that leverage visual-language pretraining and instruction tuning [8] [9]. These models have demonstrated notable effectiveness in chart understanding tasks such as ChartQA [10] and chart-to-code translation [11], benefiting from scalable and transferable training paradigms. Nevertheless, current methods face key limitations: 1) poor generalization to complex charts such as grouped bar or multi-series plots [8]; 2) high GPU memory requirements due to full fine-tuning, often exceeding 15GB [12], as all parameters require storing gradients and optimizer states; 3) monolithic model designs that lack modularity, limiting both interpretability and adaptability [13]. These issues necessitate a balanced approach to specialization and efficiency for structurally diverse inputs.

To address these challenges, this paper proposes Chart-to-Code Mixture of Low-rank Adaptation (C2C-MoLA), a multimodal framework that integrates Mixture of Experts (MoE) with Low-Rank Adaptation (LoRA) for chart-to-code generation. The framework is designed to handle structurally diverse charts by assigning inputs adaptively to specialized expert modules and fine-tuning large models efficiently under constrained computational budgets. To ensure stable and scalable training, we introduce a tailored optimization strategy for modular architectures. C2C-MoLA leverages multimodal inputs including visual chart images and chart-type metadata to generate syntactically correct and semantically faithful code, improving generalization, memory efficiency, and modularity over existing approaches.

Specifically, C2C-MoLA uses a structure-aware routing mechanism to assign inputs to eight domain-specialized experts via sparse gating, guided by a learnable complexity metric based on chart type and visual element density. To promote balanced expert utilization and prevent routing collapse, it injects Gaussian noise into gating scores and applies an auxiliary load-balancing constraint. LoRA enables efficient adaptation by applying low-rank updates to attention layers. Training further integrates syntax-aware and semantic reconstruction losses, KL-based expert regularization, and memory-efficient scheduling to support stable optimization of the architecture. Together, these components enable accurate and efficient chart-to-code generation. The main contributions of this paper include:

- Proposed C2C-MoLA, a multimodal chart-to-code generation framework that integrates MoE and LoRA to address challenges in generalization, efficiency, and modularity.

* Corresponding author: Jingyu Zhang.

979-8-3315-6653-1/25/$31.00 ©2025 IEEE
DOI 10.1109/APSEC66846.2025.00061

- Designed a complexity-aware routing strategy based on a learnable structural metric, enabling adaptive expert assignment for diverse chart types.
- Developed a training strategy that combines syntax- and semantics-aware objectives, expert regularization, and memory-efficient scheduling to support scalable optimization.
- Demonstrated consistent performance improvements over strong baselines, with ablation studies validating the contribution of each component.

II. Background and Related Work

A. Chart Understanding and Analysis

Chart understanding is fundamental to automated visual data interpretation, particularly in chart-to-code generation systems. The process begins with chart type recognition, which involves identifying categories such as bar, line, pie, and scatter charts, along with their structural layouts [14]. This recognition serves as the semantic anchor for downstream tasks [15]. Traditional methods used Convolutional neural networks to extract spatial features for classification [16], but they often struggled with long-range dependencies and complex element relationships, such as overlapping legends in grouped bar charts [17]. These limitations led to the adoption of transformer-based models, where self-attention mechanisms dynamically focus on relevant visual regions and significantly improve recognition for intricate layouts [18].

Beyond classification, precise element extraction is critical. Charts consist of components such as axes, legends, titles, labels, and data points, which define the semantic structure [19]. Early techniques relied on image processing methods such as edge detection [20] and segmentation [21], but these often struggled with ambiguous or overlapping elements. Modern deep learning approaches that combine convolutional networks for local feature extraction with transformers for contextual reasoning have shown superior performance in detection and localization accuracy [22]. These methods produce structured inputs for code generation, enabling faithful reconstruction using libraries such as Matplotlib [23] or Plotly [24].

As visualizations grow more complex, multi-modal chart understanding becomes increasingly important. Most charts embed textual elements such as axis labels and annotations within visual structures, requiring modeling of layout and semantics [25]. Vision-language architectures, including dual-attention transformers, align spatial distributions with textual cues [26]. For example, when interpreting a bar chart, a model must align spatial bar distributions with corresponding text labels to produce accurate executable code.

Ultimately, effective chart understanding requires unified modeling of both numerical distributions and textual annotations, as charts often embed semantic labels directly within visual layouts [14]. This demands not only accurate visual recognition but also cross-modal reasoning to align visual elements with their corresponding text. Recent advances further enhance this paradigm by introducing component-specific attention mechanisms that explicitly focus on key chart elements such as axes, labels, and legends [27]. These advances form a solid foundation for modular and multi-modal systems capable of handling the complexity of chart-to-code generation.

B. Vision-to-Code Generation

Automated code generation from visual inputs has emerged as a pivotal machine learning domain, driven by demands for automating web interface development and data visualization workflows [28]. This field spans diverse applications including GUI to HTML/CSS conversion [29], flowchart to program translation [30], and crucially chart to executable code generation. These tasks require models to jointly interpret visual layouts and semantic functions of elements, ensuring that the generated code accurately reflects the intended behavior.

Within chart-to-code generation, methodologies have progressed through several phases, including template-based methods, machine learning approaches, and multimodal fusion techniques. Early template-based methods matched predefined code snippets to specific chart structures [3]; while efficient for simple charts, they lacked flexibility for novel designs. Subsequently, machine-learning methods leveraged CNNs and transformers to recognize chart elements such as axes and legends, mapping them to code constructs [5]; although more capable, these models often struggle with intricate data relationships and dynamic interactivity [6]. Most recently, multimodal fusion techniques have sought to align visual and textual elements, yet still face generalization challenges across diverse chart types, due to limited handling of complex layouts, weak visual-semantic alignment, high fine-tuning costs, and poor adaptability to novel patterns [13].

As a prominent representative of recent multimodal approaches, large language models (LLMs) have significantly advanced code generation capabilities. Systems such as GPT-4.5 and Claude 3 Opus support long-context reasoning and multi-turn synthesis, while open-source alternatives like Code LLaMA [31], DeepSeek-Coder [32], and Qwen-Code [33] offer greater flexibility through fine-tuning and local deployment. Despite these gains, current LLMs still struggle with domain-specific generalization, particularly in tasks involving structured or multimodal inputs. AgentCoder [34], through multi-agent collaboration, and WizardCoder [35], via instruction tuning, show improved alignment with user intent, yet remain incapable of handling layout-sensitive or visually grounded chart generation tasks.

Recent research addresses these through modular architectures enabling adaptive reasoning. Particularly promising is the Mixture of Experts (MoE) framework, which achieves specialization while maintaining efficiency through sparse activation [36]. This naturally extends to multimodal code generation where distinct experts can handle varied visual textual patterns.

C. Mixture-of-Experts and Routing

The Mixture of Experts architecture represents a modular deep learning paradigm that achieves scalable modeling through dynamic input routing to specialized subnetworks

TABLE I: Chart-to-Code Model Comparison

Method	Routing Awareness	Structural Metric	LoRA-MoE Synergy
ChartCoder [11]	Static	×	×
ChartMoE [10]	Static	×	×
Dual-Preference [9]	×	×	×
Route-to-Reason [41]	Dynamic	×	×
C2C-MoLA (Ours)	**Dynamic**	✓	✓

[37]. To address the structural diversity and semantic complexity of chart inputs, MoE has gained increasing attention for its scalability and adaptability. Unlike conventional models that activate all parameters for every input, MoE selectively routes each instance to a small subset of experts using a gating function [36]. This sparse activation preserves model capacity while reducing computational cost, making it well-suited for heterogeneous tasks like chart-to-code generation, where different chart types demand tailored processing strategies.

Each expert in an MoE model is trained to specialize in distinct input characteristics, such as spatial layouts or annotation styles [38], enabling more accurate pattern modeling and faster convergence. However, expert overuse and underutilization often lead to load imbalance, which degrades model capacity and convergence stability [39]. Auxiliary routing losses and load-balancing regularization are commonly used to address this [40], encouraging equitable expert utilization and improving overall training efficiency.

Despite these architectural strengths, the effectiveness of MoE hinges critically on the design of routing mechanisms. Prior approaches often rely on fixed or hand-crafted rules. For instance, ChartMoE statically assigns experts based on chart type [10], providing a coarse-grained partition that overlooks intra-class variation. ChartCoder [11] employs multimodal routing but depends on hardcoded visual features, lacking adaptability to unseen structural patterns. The Dual-Preference model [9] optimizes generation through iterative tuning, yet disregards input structure in routing. Route-to-Reason (RTR) [41] introduces dynamic expert selection under computation budgets, but makes routing decisions based on latent token-level signals rather than interpretable chart structures.

A summary of these methods is shown in Tab. I, comparing their routing design, structural awareness, and integration of MoE with parameter-efficient adaptation. In contrast, our proposed C2C-MoLA introduces a dynamic routing module that leverages interpretable structural cues for fine-grained expert selection and enables seamless synergy between LoRA-based adaptation and MoE specialization.

D. Parameter-Efficient Fine-Tuning

Parameter-efficient fine-tuning has emerged as a critical strategy for adapting large pretrained models to downstream tasks while minimizing retraining costs [42]. Among various methods, Low-Rank Adaptation (LoRA) has demonstrated particular effectiveness through low rank matrix decomposition of weight updates. Rather than modifying the entire parameter space, LoRA introduces lightweight trainable matrices $\Delta W = AB$, with $B \in \mathbb{R}^{r\times d}$, $A \in \mathbb{R}^{d\times r}$, and $r \ll d$, capturing task-specific knowledge through minimal updates [43]. This approach drastically reduces computation and memory demands, making it ideal for multimodal tasks like chart-to-code generation where simultaneous visual-textual processing requires efficient resource utilization [44].

When compared to alternative methods, LoRA provides a superior balance between efficiency and expressiveness. Prefix tuning optimizes soft prompts at the input level but may underfit complex internal representations [45]. Adapters inject additional neural modules between layers, increasing flexibility at the cost of expanded model size and training time [46]. BitFit updates only bias terms to achieve maximal efficiency but struggles to capture nuanced patterns [47]. In contrast, LoRA directly modifies the weight space through low-rank projections, enabling efficient adaptation without sacrificing model capacity [48]. This characteristic makes it particularly advantageous for specialized applications demanding both precision and scalability.

LoRA offers not only parameter efficiency but also improved memory usage and faster convergence, enabling large models to be fine-tuned under limited GPU resources. This makes it especially suitable for lightweight and scalable real-time applications. When combined with LoRA, Mixture-of-Experts (MoE) models can retain adaptability even under constrained computational budgets [49]. Recent developments such as LLaMA-Adapter showcase LoRA's effectiveness in vision-language tasks [50], while MoRA [51] and Little by Little [52] attempt to integrate sparse activation with efficient tuning. However, these approaches often rely on static expert assignment or manually designed adapter placement, limiting their flexibility in handling complex or diverse input structures.

These limitations motivate C2C-MoLA's design, which integrates dynamic expert routing with LoRA adaptation, thereby enabling input-aware specialization while maintaining computational scalability. This design philosophy aims to bridge the gap between general-purpose multimodal large models and the specialized demands of chart code generation.

III. PROPOSAL

To address three core challenges including poor generalization, high GPU usage, and monolithic design, we propose C2C-MoLA, a multi-stage framework that integrates adaptive expert routing and parameter-efficient adaptation. The MoE-based router enables input-aware specialization across diverse chart types, improving generalization and modularity, while LoRA reduces memory overhead during fine-tuning.

Fig. 1 illustrates the pipeline, which starts with a chart image and metadata such as chart type and element count. This input is encoded by a dual-stream vision backbone (*DeepSeek-Coder*) combining convolutional and transformer layers to extract hierarchical visual tokens F_{vis}. These tokens are routed through a complexity-aware MoE module, where routing is guided by a learned structural complexity metric. The top-2 specialized experts are selected based on this score. Their

outputs are then fused and injected with LoRA-based low-rank updates (applied to attention projections W_q, W_k, W_v), and passed to an autoregressive decoder (*Qwen2.5-Coder*). A cross-modal attention mechanism aligns the visual features with the generated code tokens to produce executable Python code. Finally, semantic fidelity is validated by rendering the output and measuring visual similarity with the input chart validated using an IoU threshold, where $\text{IoU}(I, I') \geq \tau$.

To summarize, this formulation directly addresses three key challenges in chart-to-code generation. First, it improves modality alignment between the heterogeneous input x and output sequence S by enforcing the semantic consistency constraint (see Sec. III-A). Second, it supports scalable learning via complexity-aware expert routing and LoRA-based parameter adaptation (see Sec. III-B and III-C). Third, it balances semantic fidelity and syntactic correctness through the joint optimization of semantic rendering accuracy and a token-level syntax loss (see Sec. III-D).

A. Problem Definition

The chart-to-code generation task is formally defined as learning a mapping:

$$f : \mathbb{R}^{H\times W\times C} \to \mathcal{Y} \tag{1}$$

where the input is a chart image $I \in \mathbb{R}^{H\times W\times C}$, and the output is a token sequence $S = [s_1, s_2, \ldots, s_n] \in \mathcal{Y}$, representing executable Python code.

The output must be both syntactically correct (i.e., valid Python code), and semantically faithful. That is, the visualization rendered by executing S must align with the original chart image I.

Semantic fidelity is formalized through a validity constraint:

$$C_{\text{semantics}} = \{S \,|\, \text{execute}(S) \to I' \text{ and } \text{IoU}(I, I') \geq \tau\} \tag{2}$$

where $\text{IoU}(I, I')$ computes the Intersection-over-Union similarity between original and rendered charts, with $\tau \in [0.75, 0.90]$ as the fidelity threshold [53]. The set $C_{\text{semantics}}$ represents all code sequences S that produce renderings I' semantically consistent with the input chart I.

To enable dynamic expert selection in MoE (Sec. III-B), a learnable complexity metric is defined as:

$$c(\mathbf{x}) = \alpha \cdot \underbrace{N_{\text{elem}}(\mathbf{x})}_{\text{element count}} + \beta \cdot \underbrace{\psi_{\text{type}}(\mathbf{x})}_{\text{type complexity}} \tag{3}$$

where $\mathbf{x} = \text{VisionEncoder}(I)$ is the visual feature extracted from the input chart, $N_{\text{elem}}(\mathbf{x})$ counts the number of visual elements (e.g., bars, ticks, legends), and $\psi_{\text{type}}(\mathbf{x})$ estimates the inherent complexity of the chart type. Scalars α and β are learnable weights that adaptively balance the two components. This metric serves as the basis for dynamic expert selection, detailed in Sec. III-B.

B. MoE-Enhanced Architecture Design

As shown in Fig. 1, the Mixture of Experts (MoE) module in C2C-MoLA integrates two key components: a complexity-aware routing mechanism and a domain-specialized expert pool. The routing mechanism assigns visual tokens to the top-2 experts, selected from the expert pool based on a learned complexity metric $c(x)$, which considers both the number of visual elements and chart type difficulty. As summarized in Tab. II , expert allocation follows the chart-type distribution in the training data, with more frequent chart types (e.g., bar and line charts) receiving greater specialization. This ensures that the model can adapt to both common and rare chart types while maintaining computational efficiency and semantic fidelity.

TABLE II: Domain-Specialized Expert Configuration

Chart Type	Experts	Specialization	Ratio
Bar Charts	3	Basic, Stacked, Grouped	31.3%
Line Charts	2	Single-Series, Multi-Series	26.8%
Scatter Plots	1	Correlation Visualization	17.9%
Pie Charts	1	Proportional Representation	13.4%
Complex Charts	1	Hybrid or Atypical Layouts	8.9%

* Ratio indicates the percentage of training samples for each chart type.

1) Base Model Architecture: The dual-stream design comprises a vision encoder and a code generator. The vision encoder, based on the DeepSeek-Coder backbone combining CNN and Transformer layers, transforms the input image $I \in \mathbb{R}^{H\times W\times C}$ into a sequence of visual tokens:

$$F_{\text{vis}} = \text{VisionEncoder}(I) \in \mathbb{R}^{M\times d} \tag{4}$$

where M denotes the number of tokens and d is the token dimension.

The code generator, implemented using the Qwen2.5-Coder decoder, autoregressively generates code tokens $\{s_n\}_{n=1}^{N}$. To bridge modalities, a cross-modal attention mechanism (see Fig. 1, black arrows) aligns the visual features with the generated code tokens. At each decoding step n, attention scores between the current token s_n and each visual token f_m are computed as:

$$\text{score}(s_n, f_m) = \text{Linear}\left([\text{PE}(s_n) \oplus \text{PE}(f_m)]\right) \tag{5}$$

where $\text{PE}(\cdot)$ denotes positional encoding and $\oplus$ represents vector concatenation.

These scores are normalized using softmax to produce the attention weights:

$$A_{nm} = \frac{\exp(\text{score}(s_n, f_m))}{\sum_{m'=1}^{M} \exp(\text{score}(s_n, f_{m'}))} \tag{6}$$

The attended context vector $\hat{f}_n$ is then computed as a weighted sum of visual tokens:

$$\hat{f}_n = \sum_{m=1}^{M} A_{nm} \cdot f_m \tag{7}$$

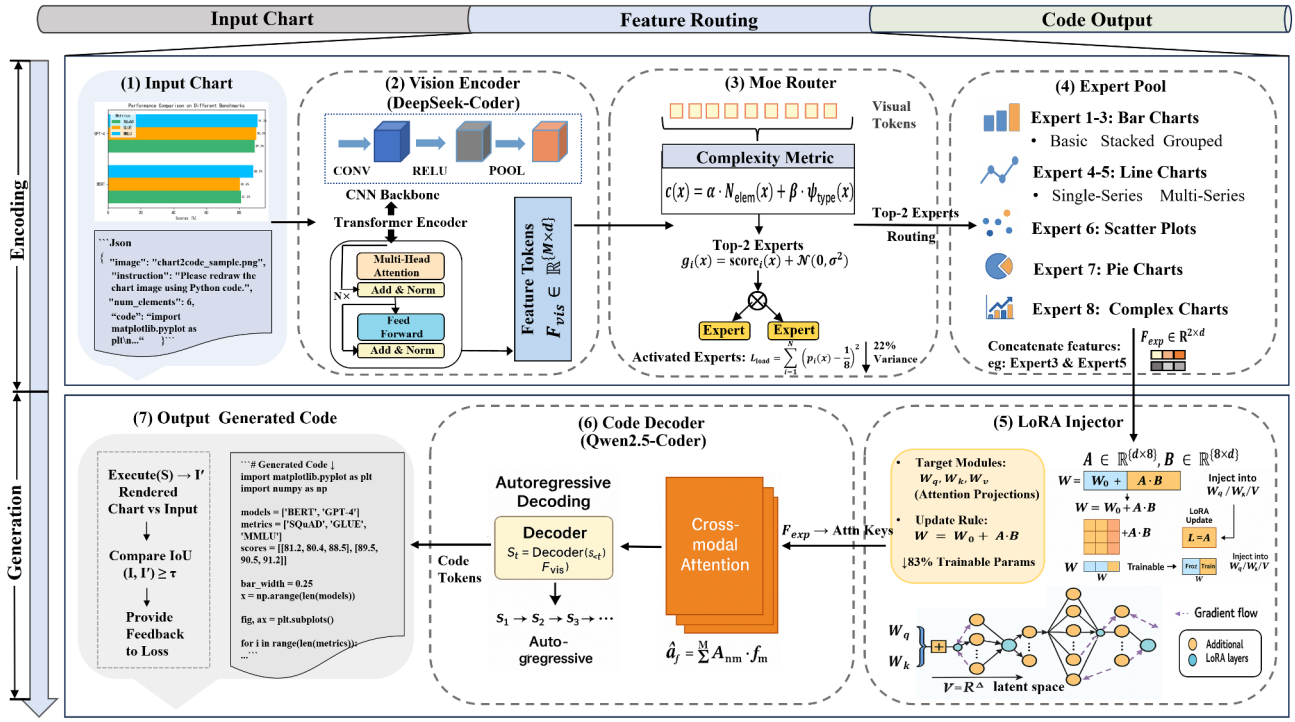

Fig. 1: C2C-MoLA: Chart-to-Code Generation Framework

This mechanism enables the decoder to focus on relevant visual regions when generating each token, ensuring semantic alignment between image content and code output.

Each expert E_i is a fine-tuned FFN optimized for its domain. To support adaptive routing, the expert selection process is guided by a complexity-aware reweighting mechanism. Given input features x, a gating network computes initial expert probabilities $p_i(x)$, which are then adjusted based on a structural complexity score $c(x)$ using temperature-scaled softmax:

$$p_i'(x) = \frac{p_i(x) \cdot \exp\left(c(x)/T\right)}{\sum_j p_j(x) \cdot \exp\left(c(x)/T\right)} \tag{8}$$

Here, T denotes a softmax temperature that controls the specialization intensity in expert routing. A higher complexity score $c(x)$ amplifies expert probabilities, encouraging routing toward more capable experts; conversely, a lower $c(x)$ reduces this effect, allowing the base routing distribution to dominate.

The final output is computed as a weighted combination of expert responses:

$$y_{\text{final}} = \sum_{i=1}^{N} p_i'(x) \cdot y_i \tag{9}$$

where $y_i = E_i(x)$ is the output of the i-th expert, and $p_i'(x)$ is the adjusted routing probability. This routing strategy aligns expert capacity with task complexity, improving both precision and generalization.

2) Routing Mechanism: To enable adaptive expert selection while maintaining computational efficiency, a softmax-based sparse gating mechanism is employed, where only the top-k experts are activated for each input. For a given input feature vector x, the gating score for expert i is computed as:

$$g_i(x) = \text{score}_i(x) + \mathcal{N}(0, \sigma^2) \tag{10}$$

where $\text{score}_i(x)$ denotes the learned relevance score, and $\mathcal{N}(0, \sigma^2)$ represents Gaussian noise injected to encourage exploration and prevent expert collapse. These scores are normalized using softmax to produce the routing probabilities $p_i(x)$, and only the top-k experts are activated based on the highest probabilities.

To promote balanced expert utilization, an auxiliary load-balancing loss is introduced to penalize skewed routing distributions:

$$L_{\text{load}} = \sum_{i=1}^{N} \left(p_i(x) - \frac{1}{N} \right)^2 \tag{11}$$

where N is the total number of experts. This loss penalizes skewed routing distributions (e.g., when 80% of inputs are assigned to a single expert), promoting diversity and robustness in expert activation.

As shown in Fig. 2, the workflow of MoE's two components is divided into four stages, illustrating the routing process from input to expert selection. First, the input feature vector $x \in \mathbb{R}^{M \times d}$ is scored via the complexity metric $c(x)$ (Eq. 3). Second, a *gating network* computes expert scores with Gaussian noise (Eq. 10). Third, these scores are normalized

using *temperature-scaled softmax* (Eq. 8) to obtain routing probabilities $p_i'(x)$, from which the top-2 domain experts are selected. Finally, expert outputs are fused via *weighted sum* (Eq. 9), and the regularization term (Eq. 11) promotes balanced expert usage.

C. LoRA-Based Parameter-Efficient Adaptation

To reduce computational and memory overhead while preserving model performance, we employ a Low-Rank Adaptation (LoRA) strategy to fine-tune the MoE-enhanced chart-to-code model. As illustrated in Fig. 1, LoRA introduces a trainable low-rank update $\Delta W = AB$ to selected weight matrices W, such that:

$$W = W_0 + \Delta W = W_0 + AB \tag{12}$$

where $A \in \mathbb{R}^{d\times r}$, $B \in \mathbb{R}^{r\times d}$, and $r \ll d$. In the implementation, LoRA is applied selectively to the attention projection matrices including W_q, W_k, and W_v, while the feed-forward networks are kept frozen to minimize parameter overhead.

For stable optimization, the matrices are initialized with $A \sim \mathcal{U}(-0.01, 0.01)$ and $B = 0$, ensuring that the initial update ΔW is zero and does not perturb the pretrained weight W_0. The overall training objective incorporates a Frobenius-norm regularization to encourage low-rank expressiveness and prevent overfitting:

$$\min_r \mathcal{L}(r) = \mathcal{L}_{\text{task}} + \lambda \left(\|A\|_F^2 + \|B\|_F^2 \right) \tag{13}$$

where λ regulates the strength of regularization. To further stabilize optimization, gradient clipping, learning rate scheduling, and rank tuning are applied. Model parameters θ = {A, B} are updated using standard gradient descent:

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L} \tag{14}$$

This strategy enables efficient task-specific adaptation of large pre-trained models, achieving high performance with significantly reduced training cost.

D. Training Strategy

To support stable and scalable optimization of the modular architecture, we design a customized training optimization strategy that facilitates stable expert routing and scalable training under memory constraints. This strategy plays a key role in enabling C2C-MoLA to realize its design objectives by aligning routing behavior with learning objectives and minimizing resource overhead. The pipeline consists of three integrated components: multi-task loss composition, optimization stabilization, and distributed memory reduction.

1) Multi-Task Loss Composition: The total loss integrates a primary code generation loss and two auxiliary components: routing loss and expert utilization regularization. Formally,

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{primary}} + \lambda_2 \mathcal{L}_{\text{router}} + \lambda_3 \mathcal{L}_{\text{util}}, \tag{15}$$

The primary loss is defined as $\mathcal{L}_{\text{primary}} = \mathcal{L}_{\text{syntax}} + \mathcal{L}_{\text{semantic}}$, ensuring both syntactic validity and visual fidelity. Specifically, $\mathcal{L}_{\text{syntax}}$ is the cross-entropy loss between the predicted and ground-truth code tokens, while $\mathcal{L}_{\text{semantic}}$ penalizes outputs whose rendered charts yield an IoU score below the threshold τ.

The router loss $\mathcal{L}_{\text{router}}$ is defined as the Kullback–Leibler (KL) divergence between the predicted expert routing distribution and a uniform prior. This encourages the model to select experts meaningfully while maintaining diversity across samples.

The utilization regularization term encourages balanced usage across all experts. It is computed as:

$$\mathcal{L}_{\text{util}} = \lambda_1 \sum_{i=1}^{n} \left| u_i - \frac{1}{n} \right|^2, \tag{16}$$

where u_i denotes the usage rate of expert i, measured over every 1k training steps. We adopt $\lambda_2 = 0.7$ and $\lambda_3 = 0.3$ based on grid search optimization.

2) Optimization Pipeline: Training is performed using the AdamW optimizer with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$, along with a weight decay coefficient of 0.01. A cosine annealing learning rate schedule is applied:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left(1 + \cos\left(\frac{t}{T_{\max}} \pi \right) \right). \tag{17}$$

where $\eta_{\max} = 10^{-4}$, $\eta_{\min} = 10^{-6}$, and $T_{\max} = 50{,}000$ steps.

To stabilize training and mitigate overfitting, dropout with a probability of 0.1–0.3 is applied to feed-forward layers, and weight decay regularization is used across parameters θ. Gradient clipping is employed to limit the magnitude of updates:

$$g_i = \frac{g_i}{\max\left(1, \frac{\|g_i\|}{c}\right)}, \tag{18}$$

Additionally, input augmentation is performed by applying random chart rotations ($\pm 15^\circ$) and color jittering ($\pm 10\%$) during training. Chart images are resized and normalized, while code sequences are tokenized into semantically meaningful units to improve visual-code alignment.

3) Memory Optimization: To support efficient training of the large-scale architecture (Qwen2.5-7B with MoE), memory optimization is achieved through three techniques. First, DeepSpeed ZeRO-3 is used to partition model parameters, gradients, and optimizer states across $K = 8$ GPUs, enabling distributed training with minimal memory consumption per device. Formally, model weights and gradients are distributed as:

$$\theta = \bigcup_{k=1}^{K} \theta_k, \quad g = \frac{1}{K} \sum_k g_k, \tag{19}$$

Second, gradient checkpointing is applied at block-level granularity (one checkpoint per four layers), reducing activation memory by approximately 47%. This technique is selectively applied to core layers, excluding MoE experts to avoid unnecessary recomputation.

Third, mixed precision training is performed using BFloat16 for forward and backward passes, while maintaining optimizer

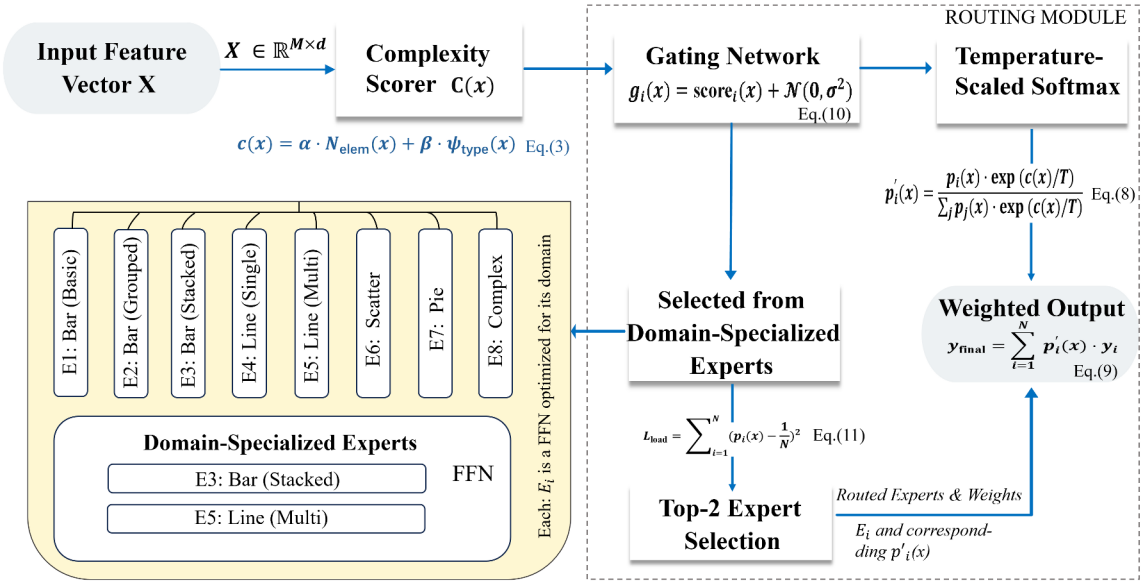

Fig. 2: Expert Routing Mechanism in C2C-MoLA

states in Float32. This reduces memory consumption by nearly 47% compared to full-precision training, without degrading convergence quality.

IV. Evaluation

In this section, the proposed C2C-MoLA framework is evaluated to validate its practical effectiveness and address the following three research questions (RQs):

- **RQ1:** Does the proposed model achieve higher generation accuracy in chart-to-code tasks, particularly when handling complex chart types?
- **RQ2:** Is the proposed model more efficient in terms of resource utilization, achieving lower memory consumption and faster convergence while maintaining high performance?
- **RQ3:** Do the design and configuration of the MoE routing mechanism and LoRA parameterization support interpretability, efficiency, and optimal trade-offs in model performance?

To address RQ1, we compare the model's execution success rate and AST-based semantic similarity across five representative chart categories (bar, line, pie, scatter, and complex charts), using both standard fine-tuning and LoRA-only approaches as baselines.

For RQ2, we analyze peak GPU memory usage, training convergence speed, and inference latency, verifying the efficiency advantages introduced by expert specialization and parameter-efficient adaptation.

To address RQ3, we conduct two complementary analyses to evaluate the design and configuration of the MoE routing mechanism and LoRA parameterization. Sec. IV-B.3 compares expert selection patterns with and without load-balancing regularization to assess whether the routing mechanism promotes balanced and potentially interpretable expert activation. Sec. IV-B.4 presents ablation studies quantifying the performance impact and computational trade-offs across different MoE configurations (expert count, token capacity, routing strategy) and LoRA settings (rank, target modules, α).

A. Settings

1) Dataset: We conduct experiments on Chart2Code-160k [53], a large-scale dataset specifically curated for chart-to-code generation research. It contains $160,000$ chart and code pairs, each consisting of a chart image and the corresponding Python visualization code. The dataset spans five representative chart types, covering a broad range of visual and structural complexity seen across chart designs.

To ensure robust evaluation, we adopt a stratified 70/15/15 split for training, validation, and testing, preserving the chart-type distribution across subsets to reduce sampling bias, as shown in Tab. III. Complex and rare layouts, including 32 variants such as stacked area and dual-axis charts, are reserved for testing to assess generalization under challenging conditions.

TABLE III: Distribution of Chart Types Across Dataset Splits

Chart Type	Training Set	Validation Set	Test Set	Complexity
Bar Chart	35,000	5,000	5,000	3.2
Line Chart	30,000	5,000	5,000	4.1
Scatter Plot	20,000	3,000	3,000	4.5
Pie Chart	15,000	2,000	3,000	2.7
Mixed Charts	10,000	2,000	2,000	Adaptive
Total	**112,000**	**24,000**	**24,000**	

* *Complexity* is computed via Eq.3 using element count and chart-type weight.

Chart images are standardized to a resolution of 512×512, and the corresponding code sequences have an average length

of 125 tokens. To improve generalization and mitigate overfitting to common patterns, we apply data augmentation at both image and code levels. For chart images, we introduce geometric transformations such as rotation, scaling, translation, and visual noise to simulate real-world variations. For code, we apply functionality-preserving modifications to variable names, function structures, and comments to increase lexical and structural diversity.

2) Baselines and Variants: To evaluate the effectiveness of C2C-MoLA, three representative baselines are compared under consistent training settings, as summarized in Tab. IV. All models are trained using the same dataset splits and preprocessing pipeline. Where applicable, learning rates, batch sizes, and stopping criteria are aligned to ensure fair comparison.

Standard Fine-Tuning: The full model is trained via backpropagation without applying MoE or LoRA. This serves as a conventional benchmark for isolating the contribution of expert routing and parameter-efficient adaptation.

LoRA-Only Adaptation: Low-rank adaptation is applied to attention projections (i.e., W_q, W_k, W_v) without using the MoE module. This setting isolates the effect of LoRA in the absence of expert routing.

Existing Chart2Code Models: ChartLLaMA and ChartCoder are included as external baselines. These transformer-based encoder-decoder models are trained on the same dataset using their original procedures. However, due to incompatible optimization settings and limited reproducibility, they are not included in the quantitative comparison.

3) Implementation Details: Experiments are conducted on NVIDIA A100 GPUs (40 GB each), connected via a high-speed InfiniBand network using the NCCL backend for distributed training. To enable scalable and memory-efficient training, we adopt DeepSpeed ZeRO-3 for parameter and optimizer state partitioning, along with gradient checkpointing to reduce activation memory. Mixed-precision training with BFloat16 is applied throughout to further reduce memory usage without compromising accuracy.

Key hyperparameters, including learning rate, optimizer, dropout, expert configuration, and routing thresholds, are summarized in Tab. V. These values were selected via grid search to optimize the interaction between LoRA adaptation and MoE routing, ensuring stable convergence and efficient expert utilization. The final MoE and LoRA settings, determined based on ablation results (Sec. IV-B.4), include 8 experts with a token capacity of 32 and top-2 routing for MoE, and a LoRA rank of 8 applied to both attention and MLP modules with a scaling factor $\alpha = 16$.

4) Evaluation Metrics: To comprehensively assess model performance, we adopt metrics spanning three key dimensions: code quality, computational efficiency, and visual-semantic alignment, as summarized in Tab. VI.

Code quality reflects the syntactic correctness, functional validity, and structural fidelity of the generated code. Outputs are evaluated based on their conformity to Python grammar and their ability to render correctly [54]. A generation is considered successful if its rendered chart matches the input with an IoU ≥ 0.85. Formally, the success rate is defined as:

$$\text{Success Rate} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\left\{\text{IoU}(I_i, I_i') \geq \tau\right\} \tag{20}$$

where $\tau = 0.85$ and $\mathbf{1}\{\cdot\}$ denotes the indicator function.

Computational efficiency covers both training and inference resource usage, including peak GPU memory, convergence time (in epochs), and inference latency per chart. Special attention is given to the overhead introduced by the MoE routing mechanism.

Visual-semantic alignment is implicitly assessed through the IoU-based success rate, which reflects how well the generated code preserves the structural and visual characteristics of the input chart [8].

B. Results

1) Overall Performance: We evaluate the overall performance of the C2C-MoLA framework across five chart types: bar, line, pie, scatter, and complex charts. Evaluation is based on success rate (rendered IoU ≥ 0.85) and visual alignment with the input chart, reflecting both syntactic correctness and semantic fidelity.

As shown in Tab. VII, C2C-MoLA consistently outperforms both standard fine-tuning and LoRA-only baselines across all chart types. Notably, it achieves success rates of 92% on bar charts and 91% on scatter plots, outperforming the baselines by 7–10%. Complex charts, which involve multi-series or overlapping elements, benefit most from adaptive expert routing, achieving a 17% improvement over standard fine-tuning.

To assess statistical significance, we conduct paired t-tests comparing the C2C-MoLA model with each baseline. The results confirm that the accuracy improvements are statistically significant ($p < 0.05$) across all chart types, indicating that the performance gains stem from architectural innovations rather than random variation.

2) Efficiency Metrics: We evaluate the efficiency of C2C-MoLA against standard fine-tuning and LoRA-only baselines across three critical dimensions: peak GPU memory consumption, convergence speed, and inference latency including routing overhead. A summary of results is shown in Tab. VII.

During training, the C2C-MoLA model reduces peak GPU memory usage by approximately 18% (from 15.0GB to 12.3GB) compared to standard fine-tuning, primarily due to selective expert activation and mixed-precision computation (BFloat16). The model also reaches convergence 20% faster (4 vs. 5 epochs), likely benefiting from LoRA's low-rank updates, which support more efficient gradient propagation.

During inference, the MoE model incurs 5% routing overhead, increasing per-chart latency by 0.05 seconds compared to the baseline. This slight increase results from the expert selection mechanism introduced by the MoE architecture.

Overall, these results show that although the MoE design introduces a slight increase in inference time, it yields clear

TABLE IV: Baseline Model Comparison

Model	Memory Efficiency	Computational Overhead	Chart Type Handling	Notable Features
Standard Fine-Tuning	Low	High	Limited to simpler charts	Full parameter updates
LoRA-Only Approach	High	Moderate	Handles a variety of charts with efficiency	Low-rank adaptation
Existing Chart2Code	Moderate	High	Varies with chart complexity	Transformer-based image-to-text models

TABLE V: Training Hyperparameter Settings

Configuration	**Details**
General Training Settings	
Batch Size	4 per device
Learning Rate	1e-4 (optimized for LoRA + MoE)
Optimizer	AdamW
Scheduler	Cosine annealing
Dropout Rate	0.1
Precision	BFloat16 (mixed precision)
Gradient Checkpointing	Enabled (for memory-time trade-off)
MoE Configuration	
Number of Experts	8
Expert Capacity	32 tokens per expert
Routing Strategy	Top-2 gating
Routing Temperature T	1.0 (empirically tuned)
Routing Noise σ	0.01
Utilization Reg. λ_1	0.5
LoRA Configuration	
LoRA Rank	8
Target Modules	Attention + MLP
Scaling Factor α	16
Evaluation Threshold	
IoU Threshold τ	0.85 (fidelity criterion)

TABLE VI: Evaluation Metrics Overview

Category	Metric	Definition
Code Quality	Success Rate	Ratio of code whose rendered charts match input (IoU ≥ 0.85). Ref lects syntax and structural fidelity.
Efficiency	Peak Memory	Max GPU and gradient cost.
	Training Time	Epochs to convergence.
	Inference Latency	Per-chart generation time.
	Routing Overhead	Extra latency from MoE routing.
Visual Match	IoU Match	Measures structural alignment between input and rendered charts via IoU threshold.

gains in training efficiency by reducing memory usage and accelerating convergence, making it well suited for deployment under constrained computational resources. As shown in Fig. 3, this C2C-MoLA model not only achieves faster convergence, but also demonstrates more stable training compared to LoRA-only and standard baselines.

3) Expert Utilization: To assess specialization and load balancing in the MoE architecture, we analyze expert selection

TABLE VII: Comparison of Accuracy and Resource Efficiency

Metric	MoE	Standard	LoRA	Δ vs.Standard
Success rate(%)				
Bar Chart	**92**	85	88	+7
Line Chart	**90**	80	85	+10
Pie Chart	**89**	75	80	+14
Scatter Plot	**91**	80	84	+11
Complex Charts	**87**	70	76	+17
Training				
Peak Memory (GB)	**12.3**	15.0	13.5	4.84
Epochs	**4**	5	**4**	1
Convergence Time (h)	**4**	5	4.5	-
Inference				
Latency (s)	**0.45**	0.40	0.42	+0.05s
Route Overhead (s)	**+0.05**	0	+0.02	-

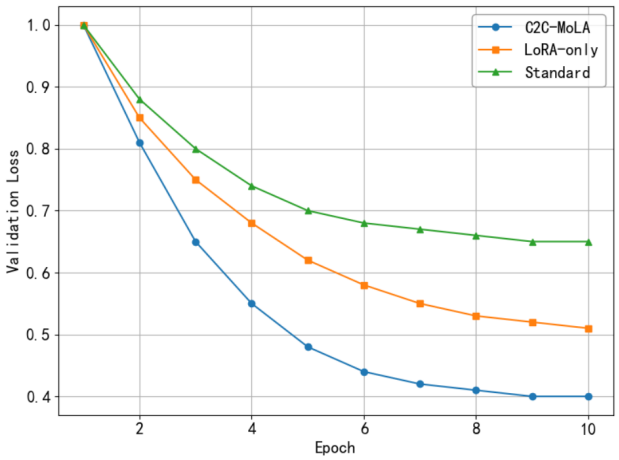

Fig. 3: Training Convergence Curves

patterns across chart types, as visualized in Fig. 4.

Before regularization, expert selection was highly skewed. For example, expert 1 handled 40% of bar chart inputs, more than double that of any other expert. Expert 3 and Expert 4 accounted for 15% of scatter plots and 10% of pie charts, respectively. Several experts, including Expert 5 through Expert 8, exhibited minimal or no activation, indicating significant load imbalance.

After applying auxiliary load balancing regularization, expert usage becomes more evenly distributed. Expert1's load for bar charts drops to 25%, while Expert3's scatter plot assignment decreases to 10%. Previously underutilized experts, such as Expert 6 and Expert 7, were activated across multiple

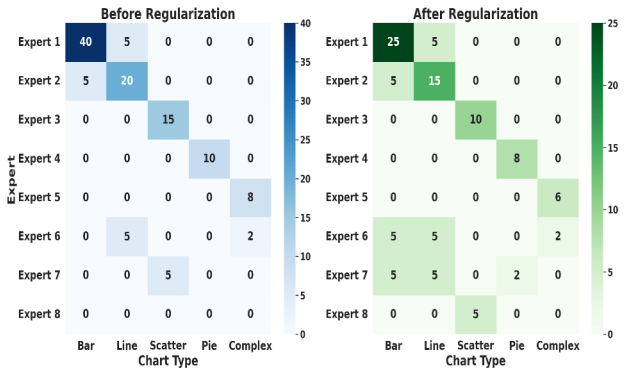

Fig. 4: Expert selection before vs. after regularization

chart types, each contributing between 2% and 5%. Expert 8, previously unused, began handling 5% of scatter plots.

This redistribution reduces the dominance of individual experts and increases overall participation. The selection matrix shows broader engagement across all eight experts and five chart types, reflecting improved task coverage and more balanced routing, which supports more stable and scalable MoE training.

4) Ablation Studies: We conduct comprehensive ablations to validate the design choices of MoE and LoRA described in Sec. III. Results in Tab. VIII identify optimal configurations that balance accuracy and resource efficiency.

a) MoE Configuration Analysis: Three key MoE parameters are evaluated: expert count, token capacity, and routing strategy. Increasing experts from 4 to 8 improves accuracy from 45% to 57% with moderate memory overhead (35%), Using 12 experts yields only a marginal gain (59%) but increases overhead to 45%. A capacity of 32 tokens per expert achieves optimal accuracy (57%), outperforming 16 tokens (43%) with a 22% memory increase. Probabilistic routing delivers better accuracy (57% vs. 52%) than top-k, albeit with slightly higher overhead (25% vs. 20%).

b) LoRA Configuration Analysis: We analyze the impact of LoRA rank, target modules, and alpha. Rank-8 achieves a strong balance between performance and training cost, improving accuracy to 58%, compared to 48% for rank-4. Rank-16 offers a marginal gain (60%) but incurs significantly higher memory and training time. Targeting both attention and output modules yields the highest accuracy (60%), surpassing attention-only (52%) and output-only (58%) settings. For alpha, a value of 32 achieves the best trade-off, reaching 58% accuracy with moderate overfitting risk (8%) and high training stability (95%). Higher alpha values (e.g., 64) introduce greater overfitting and instability.

Ablation results show diminishing returns beyond 8 experts or LoRA rank-8, accompanied by increased memory usage, overhead, or overfitting. The configuration with 8 experts, 32-token capacity, and probabilistic routing, combined with LoRA rank-8 (targeting both attention and output, alpha = 32), yields competitive accuracy with manageable resource requirements. This setup provides a balanced option for practical use.

C. Discussions

1) Generation Accuracy and Adaptability: The proposed C2C-MoLA model demonstrates consistent improvements in generation accuracy across diverse chart types, particularly under high visual or structural complexity. The dynamic routing mechanism plays a critical role in enabling specialized expert activation, enabling the model to better capture chart semantics and graphical nuances. Compared to both standard fine-tuning and LoRA-only baselines, the model achieves statistically significant performance gains, confirmed through paired t-tests. In bar charts, scatter plots, and complex visualizations, accuracy improvements range from 7% to 17%, accompanied by higher execution success rates and improved AST similarity. These findings highlight the model's ability to generalize across diverse visual patterns while preserving semantic fidelity in code generation.

2) Resource Efficiency and Scalability: The combination of MoE and LoRA leads to notable resource savings during training. Through selective expert activation and mixed-precision computation, peak GPU memory usage is reduced by approximately 18% (12.3 GB vs. 15.0 GB). Training convergence is also accelerated, requiring 20% fewer epochs compared to standard fine-tuning. While inference latency increases slightly by 0.05 seconds per chart (5% overhead), the added cost is largely offset by faster training and improved expert utilization. These results demonstrate favorable trade-offs between performance and efficiency, supporting deployment in resource-constrained or latency-sensitive environments.

3) Component Effectiveness and Design Trade-offs: Component-level evaluations validate the effectiveness of the MoE and LoRA design choices. Expert selection analysis confirms that routing behavior varies with chart type and complexity, while auxiliary load balancing promotes more uniform expert utilization. Ablation studies reveal that scaling beyond 8 experts or LoRA rank 8 yields diminishing accuracy improvements (less than 2%) at the cost of significantly increased resource usage. A token capacity of 32 and probabilistic routing provide the best trade-off between performance and overhead. For LoRA, adapting both attention and output layers with rank-8 and $\alpha = 32$ provides a strong balance between accuracy and overfitting risk.

Despite the additional inference overhead introduced by routing, latency remains acceptable for practical use (0.45s per chart), supporting the feasibility of real-world deployment. Overall, the combined MoE-LoRA architecture delivers interpretable, efficient, and accurate code generation for diverse chart inputs.

D. Validity Analysis

Internal validity. Our experiments are conducted on the Chart2Code160k dataset, which is synthetically generated. Although the dataset covers a broad range of chart types and structural variations, it may not fully capture the complexity and noise present in real-world settings, such as ambiguous

TABLE VIII: Ablation Results for MoE and LoRA

Parameters	Ours MoE								LoRA								
	Experts			Capacity			Routing		Rank			Alpha			Target Module		
	4	8	12	16	32	64	Top-k	Prob.	4	8	16	16	32	64	Attn	Out	Attn+Out
Accuracy (%)	45	57	59	43	57	59	52	57	48	58	60	55	58	60	52	58	60
Peak Memory (GB)	-	-	-	15	22	30	-	-	8	12	20	10	12	16	-	-	-
Training Time (h)	-	-	-	10	25	20	-	-	10	15	25	12	15	18	-	-	-
Overhead (%)	25	35	45	-	-	-	20	25	-	-	-	-	-	-	-	-	-
Util. Balance (%)	70	65	60	-	-	-	75	65	-	-	-	-	-	-	-	-	-
Overfit (%)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	8	20
Stability (%)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	90	95	85

* Routing: MoE strategy (Top-k = top-k experts; Prob. = probabilistic routing). Target Module: LoRA injection scope (Attn = attention; Out = output; Attn+Out = both). Overhead: MoE routing overhead (%). Util.Balance: expert utilization balance (%). "-": not measured or not applicable.

annotations, irregular layouts, or graphical artifacts. While we employ data augmentation and stratified sampling to mitigate overfitting and enhance robustness, generalization to non-synthetic or visually noisy charts remains an open challenge.

External validity. The proposed model is primarily evaluated on charts rendered from well-structured Python code using standard visualization libraries. Its performance on out-of-distribution charts, including hand-drawn sketches, scanned documents, or infographic-style visualizations, has not yet been assessed. Moreover, the model's reliance on clean and structured layouts may limit its applicability in domains with unstructured or heterogeneous visual inputs.

Implementation validity. To ensure reproducibility and fair comparison, all models are trained under consistent settings, and hyperparameters are selected with held-out validation data. Baselines are re-run using our unified codebase.

V. Conclusion

This study presents C2C-MoLA, a MoE-enhanced chart-to-code generation framework that combines adaptive expert routing with parameter-efficient LoRA fine-tuning. Evaluated on the Chart2Code-160k benchmark, the proposed model achieves 7%–17% higher generation accuracy than standard baselines,with particularly strong gains on structurally complex charts. It also reduces peak GPU memory usage by 18% and accelerates convergence by approximately 20% compared to full fine-tuning. Ablation results confirm the effectiveness of key configurations: 8 experts with a 32-token capacity and probabilistic routing, combined with a LoRA setup using rank-8 and $\alpha = 32$ applied to both attention and output layers. These findings demonstrate that the proposed architecture achieves a practical balance between accuracy, efficiency, and extensibility, enabling scalable deployment.

Future work will proceed in three directions. First, we plan to explore expert-centric pretraining, where individual experts are initialized on domain-specific chart types (e.g., a scatter-plot expert pretrained on correlation-heavy data), to reduce routing ambiguity and improve specialization stability. Second, to enhance robustness, we aim to extend the framework to handle noisy or real-world charts, including hand-drawn and scanned inputs, through sketch-augmented training and distortion-tolerant encoders. Third, we will investigate human-in-the-loop routing, integrating reinforcement learning and user feedback (e.g., preference rankings) to improve interpretability and adaptability in interactive or high-stakes settings. These directions aim to improve the generalizability, robustness, and real-world deployability of visual-to-code generation systems.

References

[1] J. Wu, W. Gan, Z. Chen, S. Wan, and H. Lin, "Ai-generated content (aigc): A survey," *arXiv preprint arXiv:2304.06632*, 2023.

[2] F. Liu, X. Wang, W. Yao, J. Chen, K. Song, S. Cho, Y. Yacoob, and D. Yu, "Mmc: Advancing multimodal chart understanding with large-scale instruction tuning," *arXiv preprint arXiv:2311.10774*, 2023.

[3] R. Minu and K. Thyagharajan, "Semantic rule based image visual feature ontology creation," *International Journal of Automation and Computing*, vol. 11, pp. 489–499, 2014.

[4] C. Gevaert, C. Persello, F. Nex, and G. Vosselman, "A deep learning approach to dtm extraction from imagery using rule-based training labels," *ISPRS journal of photogrammetry and remote sensing*, vol. 142, pp. 106–123, 2018.

[5] K. Sviatov, N. Yarushkina, and S. Sukhov, "Data extraction of charts with hybrid deep learning model," in *Computational Science and Its Applications–ICCSA 2021: 21st International Conference, Cagliari, Italy, September 13–16, 2021, Proceedings, Part IX 21*. Springer, 2021, pp. 382–393.

[6] C. Liang, F. Xue, and Z. Li, "Automatic chart decoding system based on deep learning and image processing," in *2024 4th International Symposium on Artificial Intelligence and Intelligent Manufacturing (AIIM)*. IEEE, 2024, pp. 499–504.

[7] S. Chakraborty and B. Ray, "On multi-modal learning of editing source code," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 443–455.

[8] Y. Han, C. Zhang, X. Chen, X. Yang, Z. Wang, G. Yu, B. Fu, and H. Zhang, "Chartllama: A multimodal llm for chart understanding and generation," *arXiv preprint arXiv:2311.16483*, 2023.

[9] Z. Zhang, Y. Cao, and L. Liao, "Enhancing chart-to-code generation in multimodal large language models via iterative dual preference learning," *arXiv preprint arXiv:2504.02906*, 2025.

[10] Z. Xu, B. Qu, Y. Qi, S. Du, C. Xu, C. Yuan, and J. Guo, "Chartmoe: Mixture of diversely aligned expert connector for chart understanding," in *The Thirteenth International Conference on Learning Representations*, 2025.

[11] X. Zhao, X. Luo, Q. Shi, C. Chen, S. Wang, Z. Liu, and M. Sun, "Chartcoder: Advancing multimodal large language model for chart-to-code generation," *arXiv preprint arXiv:2501.06598*, 2025.

[12] K. Li, Y. Tian, Q. Hu, Z. Luo, Z. Huang, and J. Ma, "Mmcode: Benchmarking multimodal large language models for code generation with visually rich programming problems," *arXiv preprint arXiv:2404.09486*, 2024.

[13] Y. He, Z. Liu, J. Chen, Z. Tian, H. Liu, X. Chi, R. Liu, R. Yuan, Y. Xing, W. Wang *et al.*, "Llms meet multimodal generation and editing: A survey," *arXiv preprint arXiv:2405.19334*, 2024.

[14] K.-H. Huang, H. P. Chan, Y. R. Fung, H. Qiu, M. Zhou, S. Joty, S.-F. Chang, and H. Ji, "From pixels to insights: A survey on automatic chart understanding in the era of large foundation models," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[15] A. M. Farahani, P. Adibi, M. S. Ehsani, H.-P. Hutter, and A. Darvishy, "Automatic chart understanding: a review," *IEEE Access*, vol. 11, pp. 76 202–76 221, 2023.

[16] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, "Scaling vision transformers," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 12 104–12 113.

[17] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 10 012–10 022.

[18] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1–41, 2022.

[19] K. Davila, S. Setlur, D. Doermann, B. U. Kota, and V. Govindaraju, "Chart mining: A survey of methods for automated chart analysis," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 11, pp. 3799–3819, 2020.

[20] Z. Yu, W. Liu, Y. Zou, C. Feng, S. Ramalingam, B. Kumar, and J. Kautz, "Simultaneous edge alignment and learning," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 388–404.

[21] D. Jung, W. Kim, H. Song, J.-i. Hwang, B. Lee, B. Kim, and J. Seo, "Chartsense: Interactive data extraction from chart images," in *Proceedings of the 2017 chi conference on human factors in computing systems*, 2017, pp. 6706–6717.

[22] Y. Lu, S. Mei, F. Xu, M. Ma, and X. Wang, "Separable deep graph convolutional network integrated with cnn and prototype learning for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, 2024.

[23] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 03, pp. 90–95, 2007.

[24] C. Sievert, *Interactive web-based data visualization with R, plotly, and shiny*. Chapman and Hall/CRC, 2020.

[25] S. Xiao, S. Huang, Y. Lin, Y. Ye, and W. Zeng, "Let the chart spark: Embedding semantic context into chart with text-to-image generative model," *IEEE Transactions on Visualization and Computer Graphics*, vol. 30, no. 1, pp. 284–294, 2023.

[26] W. Wang, H. Bao, L. Dong, J. Bjorck, Z. Peng, Q. Liu, K. Aggarwal, O. K. Mohammed, S. Singhal, S. Som *et al.*, "Image as a foreign language: Beit pretraining for vision and vision-language tasks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 19 175–19 186.

[27] R. Hou, B. Bühler, T. Fütterer, E. Bozkir, P. Gerjets, U. Trautwein, and E. Kasneci, "Multimodal assessment of classroom discourse quality: A text-centered attention-based multi-task learning approach," *arXiv preprint arXiv:2505.07902*, 2025.

[28] D. de Souza Baulé, C. G. von Wangenheim, A. von Wangenheim, and J. C. Hauck, "Recent progress in automated code generation from gui images using machine learning techniques." *J. Univers. Comput. Sci.*, vol. 26, no. 9, pp. 1095–1127, 2020.

[29] M. Zhekova, N. Katrandzhiev, and V. Durev, "Automatic conversion of image design into html and css," in *International Conference on IoT Based Control Networks and Intelligent Systems*. Springer, 2023, pp. 181–195.

[30] Z. Liu, X. Hu, D. Zhou, L. Li, X. Zhang, and Y. Xiang, "Code generation from flowcharts with texts: A benchmark dataset and an approach," in *Findings of the Association for Computational Linguistics: EMNLP 2022*, 2022, pp. 6069–6077.

[31] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[32] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[33] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[34] D. Huang, J. M. Zhang, M. Luck, Q. Bu, Y. Qing, and H. Cui, "Agentcoder: Multi-agent-based code generation with iterative testing and optimisation," *arXiv preprint arXiv:2312.13010*, 2023.

[35] Z. Luo, C. Xu, P. Zhao, Q. Sun, X. Geng, W. Hu, C. Tao, J. Ma, Q. Lin, and D. Jiang, "Wizardcoder: Empowering code large language models with evol-instruct," *arXiv preprint arXiv:2306.08568*, 2023.

[36] S. Mu and S. Lin, "A comprehensive survey of mixture-of-experts: Algorithms, theory, and applications," *arXiv preprint arXiv:2503.07137*, 2025.

[37] W. Fedus, J. Dean, and B. Zoph, "A review of sparse expert models in deep learning," *arXiv preprint arXiv:2209.01667*, 2022.

[38] T. Zhong, Z. Chi, L. Gu, Y. Wang, Y. Yu, and J. Tang, "Metadmoe: Adapting to domain shift by meta-distillation from mixture-of-experts," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 243–22 257, 2022.

[39] Y. Zeng, C. Huang, Y. Mei, L. Zhang, T. Su, W. Ye, W. Shi, and S. Wang, "Efficientmoe: Optimizing mixture-of-experts model training with adaptive load balance," *IEEE Transactions on Parallel and Distributed Systems*, 2025.

[40] L. Liu, J. Gao, and W. Chen, "Sparse backpropagation for moe training," *arXiv preprint arXiv:2310.00811*, 2023.

[41] Z. Pan, K. Zhang, Y. Zhao, and Y. Han, "Route to reason: Adaptive routing for llm and reasoning strategy selection," *arXiv preprint arXiv:2505.19435*, 2025.

[42] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen *et al.*, "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nature Machine Intelligence*, vol. 5, no. 3, pp. 220–235, 2023.

[43] L. Lin, H. Fan, Z. Zhang, Y. Wang, Y. Xu, and H. Ling, "Tracking meets lora: Faster training, larger model, stronger performance," in *European Conference on Computer Vision*. Springer, 2024, pp. 300–318.

[44] L. Wang, S. Chen, L. Jiang, S. Pan, R. Cai, S. Yang, and F. Yang, "Parameter-efficient fine-tuning in large language models: a survey of methodologies," *Artificial Intelligence Review*, vol. 58, no. 8, p. 227, 2025.

[45] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," *arXiv preprint arXiv:2101.00190*, 2021.

[46] R. He, L. Liu, H. Ye, Q. Tan, B. Ding, L. Cheng, J.-W. Low, L. Bing, and L. Si, "On the effectiveness of adapter-based tuning for pretrained language model adaptation," *arXiv preprint arXiv:2106.03164*, 2021.

[47] L. Xu, H. Xie, S.-Z. J. Qin, X. Tao, and F. L. Wang, "Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment," *arXiv preprint arXiv:2312.12148*, 2023.

[48] L. Zhang, L. Zhang, S. Shi, X. Chu, and B. Li, "Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning," *arXiv preprint arXiv:2308.03303*, 2023.

[49] Z. Zhang, Y. Xia, H. Wang, D. Yang, C. Hu, X. Zhou, and D. Cheng, "Mpmoe: Memory efficient moe for pre-trained models with adaptive pipeline parallelism," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[50] R. Zhang, J. Han, C. Liu, P. Gao, A. Zhou, X. Hu, S. Yan, P. Lu, H. Li, and Y. Qiao, "Llama-adapter: Efficient fine-tuning of language models with zero-init attention," *arXiv preprint arXiv:2303.16199*, 2023.

[51] C. Tang, Y. Chen, Z. Zhang, J. Shang, W. Zhang, Y. Huang, and T. Liu, "Mor: Mixture of ranks for low-rank adaptation tuning," *arXiv preprint arXiv:2410.13408*, 2024.

[52] H. Lu, C. Zhao, J. Xue, L. Yao, K. Moore, and D. Gong, "Little by little: Continual learning via self-activated sparse mixture-of-rank adaptive learning," *arXiv preprint arXiv:2506.21035*, 2025.

[53] C. Yang, C. Shi, Y. Liu, B. Shui, J. Wang, M. Jing, L. Xu, X. Zhu, S. Li, Y. Zhang *et al.*, "Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation," *arXiv preprint arXiv:2406.09961*, 2024.

[54] M. Evtikhiev, E. Bogomolov, Y. Sokolov, and T. Bryksin, "Out of the bleu: how should we assess quality of the code generation models?" *Journal of Systems and Software*, vol. 203, p. 111741, 2023.