Lite-SAM Is Actually What You Need for Segment Everything

Jianhai Fu[1,*], Yuanjie Yu[1,2,*], Ningchuan Li[1,†], Yi Zhang[1]
Qichao Chen[1], Jianping Xiong[1], Jun Yin[2], Zhiyu Xiang[2,†]

[1] Zhejiang Dahua Technology Co., Ltd.
[2] Zhejiang University

[*] Equal contribution
[†] Corresponding author

* SnapShot from Meta AI: Segment Anything Demo

Grid Search Sampling or Two-stage methods

Time Costs:
High ↑

SegEvery Results

Motivations:

The Segment Anything model (SAM) has brought significant changes to the segmentation field with its superior performance.

However, traditional Grid Search sampling strategies or two-stage concatenation methods severely limit the performance of segment everything (SegEvery).

Goal:

An efficient end-to-end solution for the SegEvery task to reduce computational costs and redundancy.

Key Contributions:

- **LiteViT:** a lightweight CNN-Transformer encoder, enhancing accuracy with reduced parameters, ideal for limited computational environments.
- **AutoPPN:** an automated prompt proposal network, improving efficiency over grid search methods and integrating easily with SAM series algorithms.
- Validated Lite-SAM's performance through experiments, showing accelerated results on SegEvery while preserving accuracy (Fig. 1).

Method: Lite-SAM

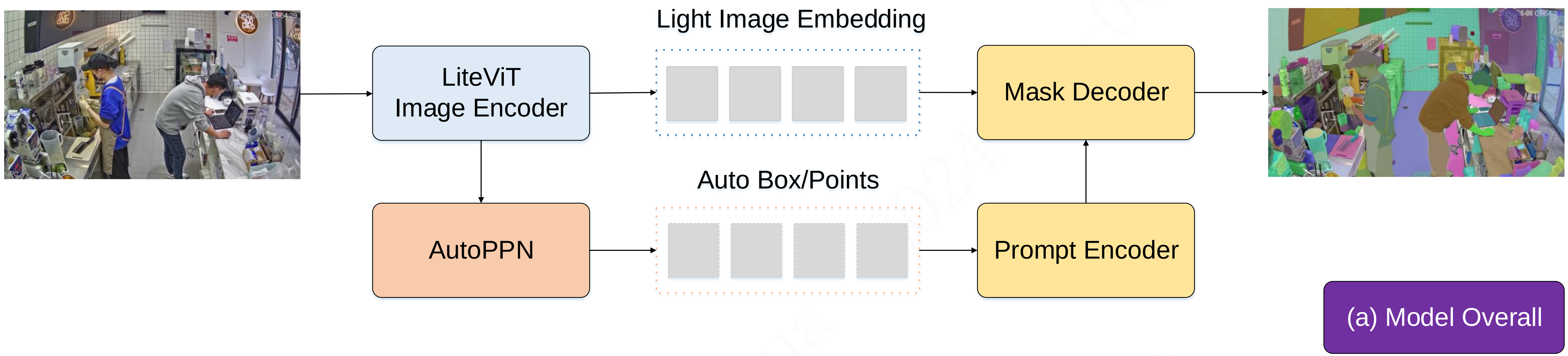
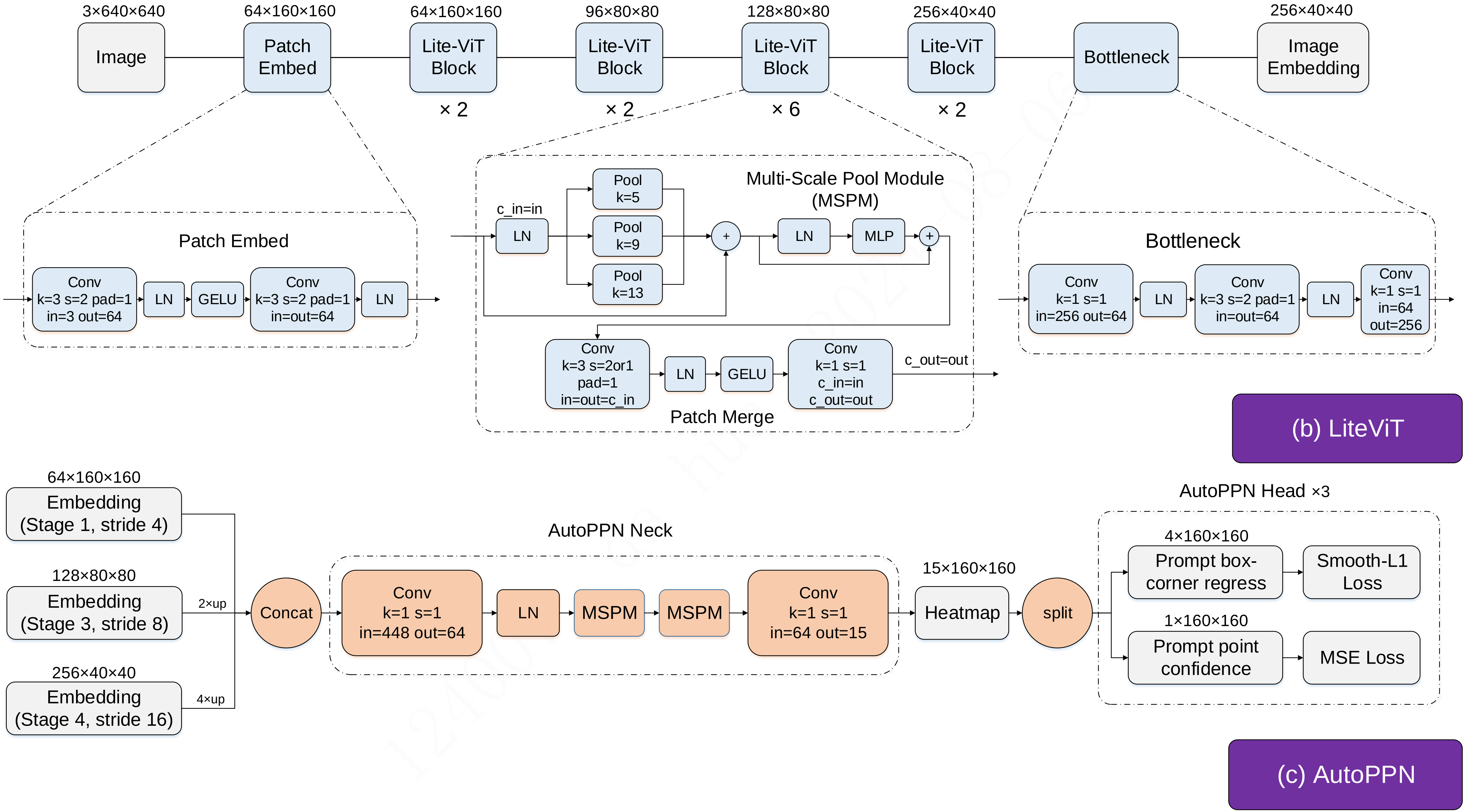


Fig. 2: (a) Overview of the proposed Lite-SAM. The architecture consists of two detachable blocks, namely the Lightwight ViT backbone (LiteViT), Automated Prompt Proposal Network (AutoPPN). (b) Macro Architecture of LiteViT. (c) Macro Architecture of AutoPPN.

We present the Lite-SAM architecture, which consists of four main components: a LiteViT encoder, an AutoPPN network, a standard prompt encoder, and a mask decoder as delineated in the SAM framework. This conguration is visualized in Fig. 2 (a).


3×640×640
Image
64×160×160
Patch Embed
64×160×160
Lite-ViT Block
× 2
96×80×80
Lite-ViT Block
× 2
128×80×80
Lite-ViT Block
× 6
256×40×40
Lite-ViT Block
× 2
Bottleneck
256×40×40
Image Embedding
Patch Embed
Conv k=3 s=2 pad=1 in=3 out=64
LN
GELU
Conv k=3 s=2 pad=1 in=out=64
LN
Multi-Scale Pool Module (MSPM)
c_in=in
LN
Pool k=5
Pool k=9
Pool k=13
+
LN
MLP
+
Conv k=3 s=2or1 pad=1 in=out=c_in
LN
GELU
Conv k=1 s=1 c_in=in c_out=out
c_out=out
Patch Merge
Bottleneck
Conv k=1 s=1 in=256 out=64
LN
Conv k=3 s=2 pad=1 in=out=64
LN
Conv k=1 s=1 in=64 out=256
(b) LiteViT
64×160×160
Embedding (Stage 1, stride 4)
128×80×80
Embedding (Stage 3, stride 8)
2×up
256×40×40
Embedding (Stage 4, stride 16)
4×up
Concat
AutoPPN Neck
Conv k=1 s=1 in=448 out=64
LN
MSPM
MSPM
Conv k=1 s=1 in=64 out=15
15×160×160
Heatmap
split
AutoPPN Head ×3
4×160×160
Prompt box-corner regress
Smooth-L1 Loss
1×160×160
Prompt point confidence
MSE Loss
(c) AutoPPN

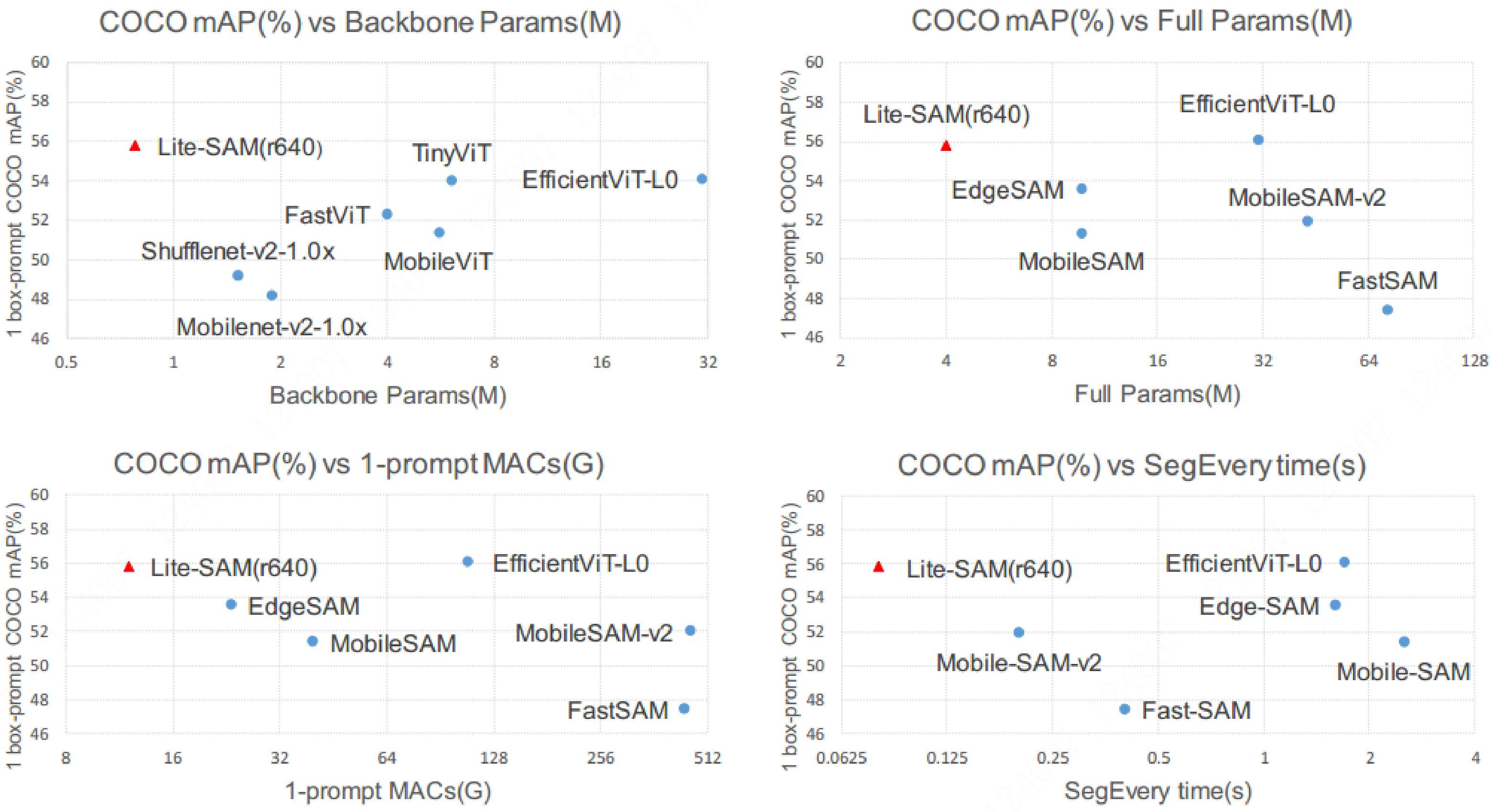


Fig. 1: The proposed Lite-SAM achieves SOTA performance in terms of Backbone Parameters (top left), Full Parameters (top right), Multiply-Accumulate Operations (bottom left), and SegEvery time (bottom right) tasks while maintaining computational efficiency. The metrics were evaluated on the zero-shot learning of the COCO dataset. Note that the comparison of backbone parameters is made against lightweight network structures (params $\leq$ 40M), with MAE not falling within this scope.

LiteViT Architecture

We developed our LiteViT image encoder, beginning with a PoolFormer-S12 baseline. The final choice based on ablation studies of LiteViT Attention block choices, as shown in Fig.2 (b), Fig. 3 and Tab. 1.

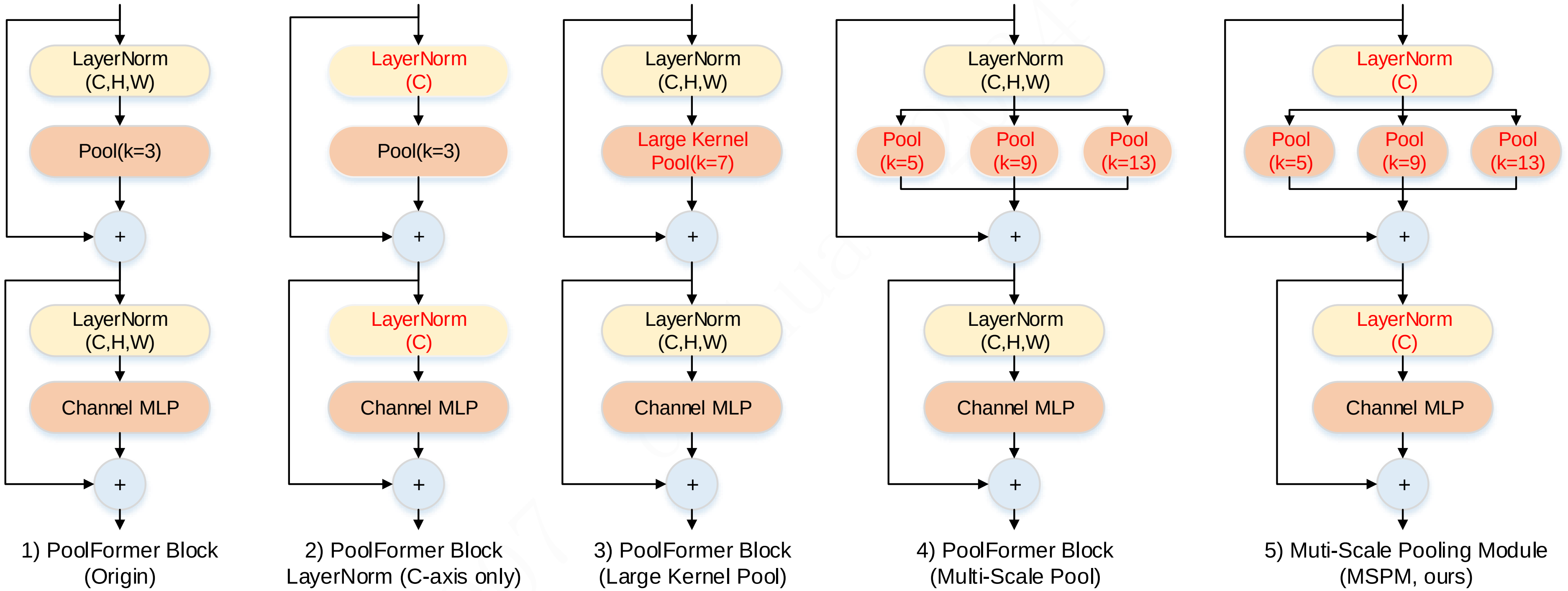


Fig. 3: Overview of architectural choice. (1) represents the original PoolFormer Block, (2),(3) and (4) show the modications to PoolFormer Block, and (5) the final version of our Multi-Scale Pooling Module.

Table 1: LiteViT Attention block Ablation Studies. All models are trained and benchmarked using the same settings described in Sec. 4.2, with unified input resolution 640×640. As a supplementary addition, we have meticulously documented the performance metrics of LiteViT, specifically its floating-point operations (FLOPs), latency, and evaluation metrics, when scaled to 2 and 3 times the parameter volume of the baseline LiteViT network. Notably, this scaling achieves impressive mAP scores of 56.9% and 58.1% for 1-box prompt segmentation on the COCO dataset, respectively. These results underscore the commendable scalability of LiteViT.

Architectual	Attention Block Choices	Params ↓ (M)	MACs ↓ (G)	Backbone Latency ↓ (ms/1-batch)	COCO 1-box prompt mAP ↑ (%)	Stages 1-4 Embedding_dims
PoolFormer-S12 (Baseline [38])	Fig. 3 (1)	11.9	45.2	30	55.1	[64, 128, 320, 512]
PoolFormer-S12-Tiny (Embedding_dims pruned)	Fig. 3 (1)	0.54	4.2	7.4	50.9	[32, 64, 96, 128]
		1.15	10.8	8.4	52.7	[64, 96, 128, 256]
	Fig. 3 (2)	1.15	10.8	8.1	53.1	
	Fig. 3 (3)	1.15	10.8	8.6	54.0	
	Fig. 3 (4)	1.16	10.9	8.7	55.2	
LiteViT (ours, Sec. 3.2)	Fig. 3 (5)	1.16 (-90%)	10.9 (-76%)	8.6 **(3.4x up ↑)**	55.3 **(+0.2% ↑)**	
LiteViT (∼2× parameters)	Fig. 3 (5)	2.19	22.3	12.4	56.9	[96, 128, 192, 384]
LiteViT (∼3× parameters)	Fig. 3 (5)	3.63	38.0	15.9	58.1	[128, 160, 256, 512]

AutoPPN

To enhance the inference performance of the SegEvery task, we introduce the AutoPPN framework, which is built on the concept of representing objects by specific points (e.g. CenterNet/CornetNet). the architecture is detailed in Fig. 2 (c).

Signicant modications / ablation studies (Tabs. 2 & 3):

- **Stem Conv to MSPM:** replacing the basic stem convolution network with a stem MSPM network, which integrates multiscale spatial information and boosts the detection recall for large-scale objects.
- **New GT and Loss:** generate GT label with distance transforms (Fig. 4) rather than bbox center approach.
- **Object Grouping:** Separate loss calculations were performed for small (relative size < 5%), medium (5% < relative size < 25%), and large size targets (relative size > 25%).
- The integration of AutoPPN leads to appreciable improvements in SegEvery time, while preserving the recall rates, As shown in Tab. 3.

Table 2: AutoPPN Ablation Studies. All models are trained and benchmarked using the same settings described in Sec. 4.2.

PPN Architectural Choices Sec. 3.3	Stem Conv → MSPM(Item 1)	New GT & Loss(Item 2)	Object Grouping(Item 3)	Mask AR@1000(%)
Baseline = Stem Conv + Focal-Loss/Smooth-L1 Loss + w/o Object Grouping	-	-	-	48.8
1 improvement strategy	✓	-	-	49.5
	-	✓	-	50.1
	-	-	✓	49.7
2 improvement strategies	✓	✓	-	51.4
	✓	-	✓	52.3
	-	✓	✓	51.1
AutoPPN (all improvement strategies)	✓	✓	✓	53.0

Table 3: Comparison of speed and accuracy acceleration of AutoPPN in SOTA models. To ensure a fair comparison, we conducted AutoPPN training on both SAM and MobileSAM using the same data and training parameters.

Model	Sampling Strategy	SegEvery Time↓ (ms)	COCO AR@1000 ↑ (%)
SAM-B [13]	Grid-Search (32 x 32)	2084	55.1
SAM-B + AutoPPN	AutoPPN(256 points)	120 (17.3x up↑)	54.7
MobileSAM [39]	Grid-Search (32 x 32)	2500	53.2
MobileSAM + AutoPPN	AutoPPN(256 points)	115 (21.7x up↑)	52.6
LiteViT(ours)	Grid-Search (32 x 32)	1320	53.4
LiteViT + AutoPPN (ours,Lite-SAM)	AutoPPN(256 points)	80 (16.5x up↑)	52.8

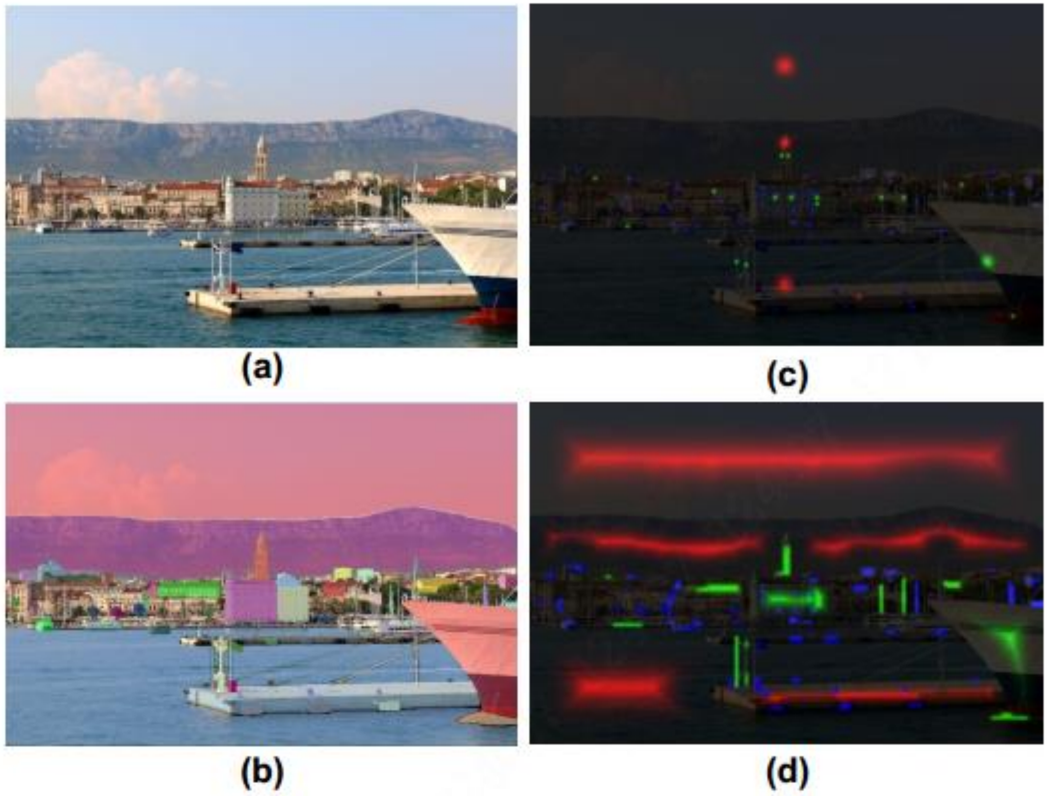


Fig. 4: We compare two methods of generating pointwise foreground/background labels within an image (sa_3196.jpg) from SA-1B [13] (a). All the masks are visualized as shown in (b). The pointwise labels generated by large, medium, small masks, are visualized with red, green and blue color, respectively. Comparing with bounding box center with gaussian kernel approach (c), distance transform approach (d) provides a more statisfactory result with less ambiguity.

Experiments

Dataset and Implementation details:

• Lite-SAM was trained on SA-1B. We selected three public datasets to assess the zero-shot capabilities of our model: MSCOCO 2017, LVIS, and BSDS500.
• We trained Lite-SAM on 128 NVIDIA A40s, with a total batch size of 256. Adam optimizer was utilized with an initial learning rate of 4e-5. The model underwent training from scratch and completed in 4 epochs, 50 hours, with using 18% of the SA-1B data, which was based on a trade-off between training time and accuracy.

Ablation study on the selection of training data size

Model	Metric & Training time (4 epochs)	Training images of SA-1B			
		1M (9%)	2M (18%)	5M (45%)	11M (100%)
SAM-B (r1024)	mAP(%)	51.9	54.4	57.4	59.0
	Hours	47	96	230	513
EfficientViT-L0-SAM (r1024)	mAP(%)	52.1	55.6	56.7	57.9
	Hours	40	83	203	427
MobileSAM (r1024)	mAP(%)	51.5	53.9	54.8	55.6
	Hours	38	75	197	402
Lite-SAM (r640)	mAP(%)	52.3	55.8	56.4	57.1
	Hours	26	50	130	272
Lite-SAM (r1024)	mAP(%)	53.9	56.5	57.6	58.2
	Hours	37	68	181	403

Comparison with SOTA Algorithms:

• Our newly developed Lite-SAM is designed as an end-to-end algorithm with a minimal parameter size of only 4.2M. Impressively, it has reduced the SegEvery runtime to a mere 80ms. This model not only demonstrates the best performance in regards to parameter size and MACs, but also in SegEvery inference time, which underlines its eciency and competitive edge.

Model Complexity, SegEvery Speed, and Mask AR@1000 metric Evaluation on COCO2017.

Model	Params ↓	MACs ↓	SegEvery Time ↓	Mask AR @1000 (%)	Sampling Strategy	Train Strategy
SAM-B	90M	371G	2.1s	55.1	GS	pretrain on MAE
SAM-L	308M	1.3T	3.3s	56.6	GS	pretrain on MAE
SAM-H	635M	2.7T	3.5s	58.7	GS	pretrain on MAE
Semantic-SAM	202M	1.4T	2.6s	55.0	GS	from scratch
EfficientViT-L0-SAM	31M	109G	1.7s	56.7	GS	from scratch
Fast-SAM	72.2M	443G	0.04s	53.3	PPOS	pretrain on YOLOv8
Mobile-SAM	9.7M	39.6G	2.5s	53.2	GS	distillation
Edge-SAM	9.7M	23.4G	1.6s	51.9	GS	distillation
Mobile-SAM-v2	43M	470G	0.13s	53.6	Object-Aware	distillation
Lite-SAM(r640)	**4.2M**	**12.7G**	**0.08s**	52.8	APPN	from scratch
Lite-SAM(r1024)	**4.2M**	32.5G	0.1s	54.1	APPN	from scratch

GS: Grid-Search (32 x 32), PPOS: Post-Process Object Selection, APPN: AutoPPN(256 points)

Comparison with SOTA lightweight backbone models on COCO.

Backbone Model	COCO 1-box (mAP)	COCO 1-box (mIoU)	COCO 1-point (mAP)	COCO 1-point (mIoU)	Seg Any time (ms)	Params (M)	MACs (G)	Input
Mobilenetv2 [10, 29]	48.2%	69.6%	23.5%	48.5%	5.1	1.89	4.0	640
Shufflenetv2 [21, 45]	49.2%	70.6%	24.2%	49.6%	5.8	1.52	5.1	640
MobileViT [24]	51.4%	72.1%	26.2%	53.8%	11.9	5.57	13.7	640
EfficientViT [20]	54.1%	73.6%	28.4%	53.1%	18.0	30.73	106.5	640
FastViT [31]	52.3%	70.0%	28.0%	48.0%	7.5	3.98	7.0	640
TinyViT [33]	54.0%	73.4%	26.5%	52.6%	17	6.07	36.6	640
LiteViT(ours, backbone)	**55.8%**	**74.8%**	**32.9%**	**55.3%**	7.9	**1.16**	10.2	640

Zero-Shot Image Segmentation Results on COCO and LVIS.

Model	MSCOCO(mIoU) ↑ 1-box (%)	MSCOCO(mIoU) ↑ 1-point (%)	LVIS(mIoU) ↑ 1-box (%)	LVIS(mIoU) ↑ 1-point (%)	MSCOCO↑ AP (%)	MSCOCO↑ AP^S (%)	MSCOCO↑ AP^M (%)	MSCOCO↑ AP^L (%)	LVIS ↑ AP (%)	LVIS ↑ AP^S (%)	LVIS ↑ AP^M (%)	LVIS ↑ AP^L (%)
SAM-B [13]	75.0	52.2	73.5	55.2	56.6	47.4	60.3	68.0	61.1	50.3	71.6	76.7
SAM-L [13]	76.4	56.8	75.0	55.8	59.4	48.8	65.3	72.1	64.9	53.5	76.1	81.9
SAM-H [13]	**76.5**	**57.4**	**75.3**	**56.4**	**59.8**	49.4	63.8	71.9	**65.2**	53.6	76.5	82.1
Semantic-SAM-L [15]	N/A	54.7	N/A	34.8	N/A							
Fast-SAM [46]	72.8	50.2	67.3	46.8	47.5	37.9	48.1	56.4	43.8	35.1	45.6	59.7
EfficientViT-L0-SAM [4]	74.5	51.3	73.1	52.9	56.1	44.3	59.7	70.8	59.8	46.8	70.2	80.1
EfficientViT-L1-SAM [4]	75.2	51.5	73.9	54.8	57.1	45.4	60.8	71.5	61.4	48.0	72.5	81.6
Mobile-SAM-v2* [40]	72.8	50.5	67.7	42.4	51.4	41.6	55.1	64.1	52.8	42.2	63.2	69.6
Mobile-SAM [39]	72.8	50.5	67.7	42.4	51.4	41.6	55.1	64.1	52.8	42.2	63.2	69.6
Edge-SAM [47]	74.0	51.9	69.4	43.8	52.5	42.7	56.0	65.3	54.1	43.5	63.9	70.7
Lite-SAM(ours, r640)	74.8	55.8	73.2	54.4	55.8	46.7	59.6	69.6	58.4	45.9	66.9	77.5
Lite-SAM(ours, r1024)	**76.3**	**56.9**	**75.7**	**57.3**	**56.5**	47.4	61.0	70.7	**60.7**	49.3	71.9	79.8

Zero-shot transfer to edge detection on BSDS500.

Method	Year	ODS	OIS	AP	R50
HED	2015	0.788	0.808	0.840	0.923
EDETR	2022	0.840	0.858	0.896	0.930
zero-shot transfer methods:					
Sobel filter	1968	0.539	-	-	-
Canny	1986	0.600	0.640	0.580	-
Felz-Hutt	2004	0.610	0.640	0.560	-
SAM-H	2023	0.768	0.786	0.794	0.928
Fast-SAM	2023	0.750	0.790	0.793	0.903
Lite-SAM(ours)	2023	0.761	0.788	0.793	0.919

Qualitative results on SegEvery Task

Thank you !

For more details, please refer to our paper:
https://arxiv.org/abs/2407.08965
For further discussion, please scan the QR code: