Supplementary Materials Organization:

A More Experimental Analysis

A.1 Base-to-new Generalization

Results. Our method can be extended to the base-to-new generalization scenario by incorporating the KNN algorithm. To accomplish this, we utilize the text embeddings of the new classes to query the training set and select the k nearest neighbors as the training data for the new class. Subsequently, we apply our proposed method to generate the classifier for the new classes using the synthesized dataset. In order to compare our approach, we select CLIP (Radford et al., 2021), CoOp (Zhou et al., 2022b), and CoCoOp (Zhou et al., 2022a).

Table 8 presents the results, which demonstrate that our approach outperforms the other methods in terms of base accuracy, new accuracy, and their harmonic mean. On average across 11 datasets, our method surpasses CLIP, CoOp, and CoCoOp by 14.62%, 1.27%, and 3.49% in terms of base accuracy. It also outperforms them by 0.31%, 11.31%, and 2.84% in terms of new accuracy, and by 7.02%, 7.06%, and 2.89% in terms of the harmonic mean. Moreover, our approach achieves the highest harmonic mean in 8 out of 11 datasets. These results clearly indicate the effectiveness of our approach in generalizing to new classes.

A.2 Robustness to Different Architectures

We further evaluate the efficacy of our proposed method across 11 datasets with varying visual architectures of CLIP. We selected two approaches for comparison: a training-required method, CoOp (Zhou et al., 2022b), and a training-free method, Tip-Adapter (Zhang et al., 2022). And these methods are trained on the 16-shot dataset. As shown in Table 9, our method yielded a substantial improvement of 17.28%, 18.20%, 16.18%, and 16.62% on average, compared to the zero-shot CLIP (Radford et al., 2021) approach, for ResNet-50, ResNet-101, ViT-B/32, and ViT-B/16 CLIP, respectively, across all 11 datasets. The results demonstrate the effectiveness of our method across different CLIP architectures.

B Experimental Details

B.1 Statistic of Datasets

Following previous work (Zhou et al., 2022a;b; Wang et al., 2023c; Huang et al., 2022; Wang et al., 2023a), we conduct experiments on 17 publicly available image classification datasets. The datasets include ImageNet (Deng et al., 2009), Caltech101 (Li et al., 2004), OxfordPets (Parkhi et al., 2012), StanfordCars (Krause et al., 2013), Flowers102 (Nilsback & Zisserman, 2008), Food101 (Bossard et al., 2014), FGVCAircraft (Maji et al., 2013), EuroSAT (Helber et al., 2019), UCF101 (Soomro et al., 2012), DTD (Cimpoi et al., 2014), SUN397 (Xiao et al., 2010), ImageNetV2 (Recht et al.,

(a) **Average over 11 datasets**

	base	new	**H**
CLIP	69.34	74.22	71.70
CoOp	82.69	63.22	71.66
CoCoOp	80.47	71.69	75.83
Ours	**83.96**	**74.53**	**78.72**

(b) ImageNet

	base	new	**H**
CLIP	72.43	68.14	70.22
CoOp	**76.47**	67.88	71.92
CoCoOp	75.98	**70.43**	**73.10**
Ours	75.95	69.79	72.74

(c) Caltech101

	base	new	**H**
CLIP	96.84	94.00	95.40
CoOp	98.00	89.81	93.73
CoCoOp	97.96	93.81	95.84
Ours	**98.04**	**94.51**	**96.24**

(d) OxfordPets

	base	new	**H**
CLIP	91.17	97.26	94.12
CoOp	93.67	95.29	94.47
CoCoOp	**95.20**	**97.69**	**96.43**
Ours	94.10	97.15	95.60

(e) StanfordCars

	base	new	**H**
CLIP	63.37	**74.89**	68.65
CoOp	78.12	60.40	68.13
CoCoOp	70.49	73.59	72.01
Ours	**78.71**	66.92	**72.34**

(f) Flowers102

	base	new	**H**
CLIP	72.08	**77.80**	74.83
CoOp	97.60	59.67	74.06
CoCoOp	94.87	71.75	81.71
Ours	**97.78**	72.46	**83.24**

(g) Food101

	base	new	**H**
CLIP	90.10	91.22	90.66
CoOp	88.33	82.26	85.19
CoCoOp	**90.70**	**91.29**	**90.99**
Ours	90.63	91.21	90.92

(h) FGVCAircraft

	base	new	**H**
CLIP	27.19	**36.29**	31.09
CoOp	40.44	22.30	28.75
CoCoOp	33.41	23.71	27.74
Ours	**45.88**	34.09	**39.12**

(i) SUN397

	base	new	**H**
CLIP	69.36	75.35	72.23
CoOp	80.60	65.89	72.51
CoCoOp	79.74	**76.86**	78.27
Ours	**81.95**	75.62	**78.65**

(j) DTD

	base	new	**H**
CLIP	53.24	**59.90**	56.37
CoOp	79.44	41.18	54.24
CoCoOp	77.01	56.00	64.85
Ours	**80.63**	59.82	**68.69**

(k) EuroSAT

	base	new	**H**
CLIP	56.48	64.05	60.03
CoOp	92.19	54.74	68.69
CoCoOp	87.49	60.04	71.21
Ours	**93.28**	**79.21**	**85.67**

(l) UCF101

	base	new	**H**
CLIP	70.53	77.50	73.85
CoOp	84.69	56.05	67.46
CoCoOp	82.33	73.45	77.64
Ours	**86.63**	**79.09**	**82.69**

Table 8: **Base-to-new generalization.** Comparison of CLIP, CoOp, CoCoOp, and our method. CoOp and CoCoOp are training-required methods, while our method is a training-free method. base and new denotes the average accuracy of base and new classes, and H denotes their harmonic mean.

Table 9: **Robustness of different architectures on 11 datasets.** The models are trained under the 16-shot setting with different visual architectures of CLIP. **Bold** denotes the highest results.

Method	**Pets**	**Flowers**	**FGVC**	**DTD**	**EuroSAT**	**Cars**	**Food**	**SUN**	**Cal**	**UCF**	**IN**	**Avg.**
ResNet-50												
zero-shot CLIP	85.77	66.14	17.28	42.32	37.56	55.61	77.31	58.52	86.29	61.46	58.18	58.77
CoOp	87.01	94.51	31.26	63.58	83.53	73.36	74.67	69.26	91.83	75.71	62.95	73.42
Tip-Adapter	88.14	89.89	29.76	60.93	70.54	66.77	77.83	66.85	90.18	70.58	62.01	70.32
Ours	**88.81**	**95.72**	**40.61**	**66.51**	**86.12**	**75.12**	**79.05**	**70.70**	**92.55**	**77.53**	**63.82**	**76.05**
ResNet-101												
zero-shot CLIP	86.75	64.03	18.42	38.59	32.59	66.23	80.53	58.96	89.78	60.96	61.62	59.86
CoOp	88.57	95.19	34.76	65.47	83.54	79.74	79.08	71.19	93.42	77.95	**66.60**	75.96
Tip-Adapter	87.23	90.77	31.51	62.37	66.45	72.96	81.31	67.96	93.01	73.53	64.41	71.96
Ours	**91.43**	**96.17**	**42.58**	**68.62**	**86.32**	**79.99**	**82.15**	**72.07**	**93.63**	**79.31**	66.33	**78.06**
ViT-B/32												
zero-shot CLIP	87.49	66.95	19.23	43.97	45.19	60.55	80.50	61.91	90.87	62.01	62.05	61.88
CoOp	88.68	94.97	33.22	65.37	83.43	76.08	78.45	72.38	**94.62**	78.66	66.85	75.70
Tip-Adapter	88.34	91.61	30.92	61.90	69.53	69.59	80.94	70.27	93.85	73.74	65.41	72.37
Ours	**91.21**	**96.16**	**41.74**	**67.63**	**87.30**	**77.55**	**81.84**	**73.60**	94.42	**80.17**	**67.00**	**78.06**
ViT-B/16												
zero-shot CLIP	89.21	71.34	24.72	44.39	47.60	65.32	86.06	62.50	92.94	66.75	66.73	65.23
CoOp	92.53	96.47	42.91	68.50	80.87	**83.09**	87.21	75.29	95.77	82.24	71.92	79.71
Tip-Adapter	91.54	94.41	39.48	65.68	76.58	75.44	86.47	71.85	95.10	77.94	70.46	76.81
Ours	**93.73**	**97.92**	**50.33**	**71.26**	**89.19**	82.63	**87.27**	**75.87**	**95.79**	**84.09**	**72.24**	**81.85**

2019), ImageNet-Sketch (Wang et al., 2019), ImageNet-A (Hendrycks et al., 2021b), ImageNet-R (Hendrycks et al., 2021a), ImageNet-LT (Liu et al., 2019), and Places-LT (Zhou et al., 2017).

Table 10: Detailed statistics of datasets used in experiments.

Dataset	# Classes	# Training	# Test	Task
OxfordPets	37	2,944	3,669	fine-grained pets recognition
Flowers102	102	4,093	2,463	fine-grained flowers recognition
FGVCAircraft	100	3,334	3,333	fine-grained aircraft recognition
DTD	47	2,820	1,692	Textural recognition
EuroSAT	10	13,500	8,100	Satellite image recognition
StanfordCars	196	6,509	8,041	Fine-grained car recognition
Food101	101	50,500	30,300	Fine-grained food recognition
Sun397	397	15,880	19,850	Scene recognition
Caltech101	100	4,128	2,465	Object recognition
UCF101	101	7,639	3,783	Action recognition
ImageNet	1,000	1.28M	50,000	Object recognition
ImageNetV2	1,000	-	10,000	Robustness of collocation
ImageNet-Sketch	1,000	-	50,889	Robustness of sketch domain
ImageNet-A	200	-	7,500	Robustness of adversarial
ImageNet-R	200	-	30,000	Robustness of rendition styles
ImageNet-LT	1,000	115,846	50,000	long-tail object recognition
Places-LT	365	62,500	7300	long-tail place recognition

B.2 Prompt Templates for Each Dataset

For the zero-shot classifier, we employ handcrafted prompts to generate the classifier weight, as proposed in CLIP (Radford et al., 2021). By default, we utilize the prompt template "a photo of {class}." for class labels, where {class} represents the name of the classes. However, for fine-grained classification datasets such as FGVCAircraft (Maji et al., 2013), we incorporate the name of the superclass or a description into the template. The prompt templates for each dataset are shown as follows.

B.3 Pseudocode

Table 11: Prompt templates for each class.

Dataset	Prompt template
Caltech101 (Li et al., 2004)	"a photo of a {class}."
OxfordPets (Parkhi et al., 2012)	"a photo of a {class}, a type of pet."
StanfordCars (Krause et al., 2013)	"a photo of a {class}."
Flowers102 (Nilsback & Zisserman, 2008)	"a photo of a {class}, a type of flower."
Food101 (Bossard et al., 2014)	"a photo of {class}, a type of food."
FGVCAircraft (Maji et al., 2013)	"a photo of a {class}, a type of aircraft."
SUN397 (Xiao et al., 2010)	"a photo of a {class}."
DTD (Cimpoi et al., 2014)	"{class} texture."
EuroSAT (Helber et al., 2019)	"a centered satellite photo of {class}."
UCF101 (Soomro et al., 2012)	"a photo of a person doing {class}."
ImageNet (Deng et al., 2009)	"a bad photo of the {class}." "a origami {class}." "a photo of the large {class}." "a {class} in a video game." "art of the {class}." "a photo of the small {class}."
ImageNet-LT (Liu et al., 2019)	"a photo of a {class}."
Places-LT (Zhou et al., 2017)	"a photo of a {class}."

Algorithm 1 Pytorch-like pseudocode for our method.

```
# Input:
# - X: (N, D) visual features from CLIP visual encoder.
# - Y: (N, ) ground-truth label for the features.
# - X_test: (M, D) test visual features from CLIP visual encoder.
# - Y_test: (M, ) ground-truth label for test features.
# - W_c: (K, D) zero-shot classifier generated by prompting.
# Output:
# - acc: test accuracy.

def hard_to_beat(X, Y, X_test, Y_test, W_c):
    # 1. Compute mean vectors for each class.
    mus = []
    for i in range(K):
        idx = torch.where(Y == i)
        mus.append(X[idx].mean(dim=0))
    mus = torch.cat(mus)

    # 2. Estimate the precision matrix using Equation (4).
    # centered features
    centered_X = torch.cat([(X[torch.where(Y == i)] - mus[i]) for i in range(K)])
    cov = torch.cov(centered_X)
    # compute the precision matrix (inverse covariance)
    inv_cov = D * torch.inv((N - 1) * cov + trace(cov) * eye(D))

    # 3. Compute weight and bias using Equation (3).
    W = mus @ inv_cov
    b = log(1 / K) - 0.5 * einsum('nd, dc, nc -> n', mus, inv_cov, mus)

    # 4. Search the hyperparameter using the validation set.
    alpha = search_hyperparam(W_c, W, b)

    # 5. Test.
    test_logits = X_test @ W_c.T + alpha * (X_test @ W.T + b)
    acc = compute_acc(test_logits, Y_test)
return acc
```