A APPENDIX

A.1 SUPPLEMENTARY THEORETICAL ANALYSIS

In Section 4, we suppose the Transformer encoder f_θ is permutation equivariant. This result was proved in Kossen et al. (2021), and for convenience, we repeat the properties, lemmas, and definitions developed in this work with language adapted to few-shot and in-context learning. Kossen et al. (2021) can be referenced for the corresponding proofs.

Definition 2. *A function $f_\theta : \mathcal{S}^n \to \mathcal{S}^n$ is permutation-equivariant if for any permutation $\pi : [1, \ldots, n] \to [1, \ldots, n]$ applied to the sequence elements of $\mathcal{S}^n$, we have for all i, $f(S_1, \ldots, S_n)[i] = f(S_{\pi^{-1}(1)}, \ldots, S_{\pi^{-1}(n)})[\pi(i)]$.*

Lemma 3. *Any function of the form $f_\theta(S_1, \ldots, S_n) = (g(S_1), \ldots, g(S_n))$ for some g is permutation-equivariant. These functions are denoted as 'element-wise operations', as they consist of the same function applied to each of element of the sequence.*

Lemma 4. *The composition of permutation-equivariant functions is permutation-equivariant.*

Lemma 5. *Let $W \in \mathbb{R}^{n \times m_1}$ and $S \in \mathbb{R}^{m_2 \times n}$. The function $S \mapsto SW$ is permutation-equivariant.*

Lemma 6. *The function $X \mapsto Att(SW^Q, SW^K, SW^V)$ is permutation-equivariant.*

Lemma 7. *The following hold:*

1. *Multihead self-attention is permutation-equivariant.*
2. *If f and g are permutation-equivariant, then the function $x \mapsto g(x) + f(x)$ is also permutation-equivariant.*
3. *The residual connection in a Transformer encoder block is permutation-equivariant.*
4. *The Transformer encoder block itself is permutation-equivariant.*

Property A.1.1. *The Transformer encoder f_θ is permutation-equivariant.*

A.2 ADDITIONAL TRAINING DETAILS

Training at Different Support Sizes. While pre-training on FS-Mol, CAMP used example sequence sizes of $\{16, 32, 64, 128, 256\}$. Similar to the training of language models or other sequence-based methods that group examples within a batch to have a similar sequence length, we compose all examples within a batch to have the same support size.

It is also the case that an initial example sequence of size $|s|$ actually encodes $|s|$ different example sequences, each with sequence length $|s|$ by changing the designation of support and query points within that sequence. For instance, we could designate the first example in the sequence as the query and all other points in the sequence as the support. Repeating this process for at every position in the sequence—so that each example in the sequence is designated as the query exactly once—yields $|s|$ example sequences from an initial example sequence. We found this protocol to be imperative to training stability and posit it serves as an implicit form of data augmentation, supplying the model with additional training examples from a leave-one-out style strategy.

As we choose a batch size of 256, the "effective" batch size for each sequence length changes. For example, let $|s| = 16$, then the effective batch size will contain $16 * 256$ =4,096 example sequences. Similarly, if $|s| = 256$, then the effective batch size will contain $256 * 256$ =65,536 example sequences. We note that this augmentation is only performed during training. Future work may explore also applying it during evaluation in a manner similar to noisy channel models (Brown et al., 1993; Koehn et al., 2003; Min et al., 2021a) to improve test-time performance. Further, as we train on a single A100 GPU and use the "base" variant ViT, GPU memory is not a limiting factor.

Nevertheless, without rebalancing batch sizes to depend on sequence length, our sequence-augmentation protocol may lead to the model overfitting to smaller support sizes. For example, suppose we instead set our batch size to 2 and train on a single dataset composed of 512 measurements. Then we have 32 different 16-size example sequences by allocating each measurement in the dataset to be used by only a single example sequence of size 16. Similarly, we would have 2 different

256-size example sequences by allocating each measurement in the dataset as belonging to only one example sequence of size 256.

As the batch size is set to 2, the model would see 16 batches composed of 16-size initial example sequences but only a single batch composed of a 256-size initial example sequence. Moreover, the per-batch loss is normalized by the user-set batch size (i.e. 2 in this example) and gradient values are clipped by their norm. Accordingly, the model's parameters are updated at a ratio of 16:1 for 16-size example sequences to 256-size example sequences. Training with respect to effectively 16x more batches at the smaller sequence length may bias the model towards smaller sequence sizes and may account for CAMP losing ground to other meta-learning baselines at large support sizes.

Training Hyperparameters. Our optimization protocol uses the Adam (Kingma & Ba, 2014) optimizer with learning rate of 5×10^{-5} with a linear warm-up schedule of 2,000 steps. Adam uses the default β_1 and β_2 parameters set by Pytorch (Paszke et al., 2019). We follow the implementation of Dosovitskiy et al. (2020) for the Transformer encoder, using the "base" variant and setting dropout to 0.2 after a very coarse hyperparameter search over #warmup steps = $\{100, 2000\}$ and dropout = $\{0, 0.05, 0.1, 0.2\}$. We use early stopping on the FS-Mol validation split, with a window-size of 10 and using the Validation cross entropy loss as our stopping criterion.