We thank the reviewers for thoughtful feedback, and for acknowledging the importance and novelty of this work. We focus on **absolute speedups** and **trade–offs** by providing additional results as further evidence of practical effectiveness.

Time and memory trade–offs of hypersolvers [All]: We agree that the number of function evaluations of the Neural ODE vector field f (NFE), while a commonly used measure in the literature, does not take into account the hypersolver cost. To address concerns related to hypersolver overheads, Fig. 1 provides additional pareto optimality plots (solution error & test acc. loss) against `MACs`, *multiply-accumulate* operations of f (for baselines) and $f + g$ (for hypersolvers). `MACs`, similar to `FLOPs`, are commonly used as an hardware–agnostic algorithmic complexity measure. We note that in the current implementation, `HyperEuler` corrections require less than 50% of the `MACs` required for an evaluation of the f network. In cases where the architecture of f is itself is deeper, the overhead of hypersolvers is reduced, further strengthening their pareto efficiency. Memory overheads are of similar magnitude, though memory is less of a concern in Neural ODE due to their smaller footprint enabled by the $O(1)$ memory adjoint sensitivity technique for training.

Absolute speedup [All]: To further contextualize the efficacy of hypersolvers, we provide absolute time and speedup plots (TITAN V, CUDA v10.2) in Fig. 2. The baselines are set to perform the minimum number of steps necessary to preserve test classification accuracy. In contrast with `MACs`, these results take into account implementation and hardware overheads. It should be noted that the relative baseline ranking can different w.r.t `MAC` plots due to implementation overheads (here `torchdiffeq`). Here, `HyperEuler` provides 8x speedup over dopri5 with exact same test accuracy.

Scope [R3]: We agree with **R3** that there exist a vast literature on speeding up simulators and solvers with neural networks, and have added the references suggested by **R3**. We do not claim to be the first to introduce the idea of offline solver pretraining, nor do we claim state–of–the–art in the general case. The core contribution is comprised of the hypersolver formulation, theoretical guarantees and training strategies tailored for performance in the context continuous models. We believe the emerging learning–based interplay between Neural ODEs and their solver (Sec. 6) to also be a valuable contribution in itself, especially given the increasing use of Neural ODEs across scientific fields.

Alternative approaches and surrogate models [R3, R4]: We argue that learning only the residual *higher–order* term, instead of the full map $\mathbf{z_k} \mapsto \mathbf{z_{k+1}}$ is advantageous, in the context of Neural ODEs, for several reasons. We notice that the residual fitting requires a substantially smaller NN compared to learning directly the solution of the ODE, which does not make explicit use of the vector field f. Th.m 1 further provides theoretical guarantees on the solver's truncation error improvement, relevant for safety–critical applications. Moreover, to the best of our knowledge, this is the first Neural ODE solver designed to learn residuals, showcasing **large speedups** (as shown above) of **practical relevance**.

CNF likelihood eval. [R1]: Hypersolvers can also be used during CNF likelihood evaluation. We verified this experimentally and obtained comparable results to CNF sampling; a discussion will be included.

Extension of Section 6 [R1, R3]: We agree that Sec. 6 represents an important component of the work and in its current form already highlights original aspects of model–solver interplay. We decided, however, to dedicate more space to the core formulation and results to provide a solid foundation for more complex hypersolver architectures.

Sensitivity to tolerance of ground-truth method [R2]: Tolerances can be regarded as hyperparameter of f itself. If insufficient, the solutions might have high *error* and lower Neural ODE task performance; regardless, the hypersolver will learn to match the residuals, ensuring the base method is able to track ground–truth solutions. Hypersolvers are only sensitive to tolerances in the sense that they represent an upper–bound on the accuracy of *hypersolved* solutions.

Generalization to different step–sizes [R4]: Generalization across step–sizes is shown via pareto plots (also Fig. 1). The integration interval is fixed to $S := [0, 1]$ for all NFEs; hence, traversing the *x–axis* corresponds to a denser discretization of S. `HyperEuler` is competitive even far from its training step size $\epsilon = 0.1$ (10 NFEs).

Hyper–hyper [R1]: This is a valuable suggestion. By progressively training an ensemble networks on increasing–order residuals one could reach a local truncation err. $\mathcal{O}(\delta_1\delta_2\cdots\delta_p\epsilon^2)$ instead of $\mathcal{O}(\delta\epsilon^{p+1})$ obtained by directly fitting the residual of a p-ord. solver. Understanding whether this is convenient in practice is certainly worth further investigations.

Clarifications: [R4] $\ell_{\cancel{\texttt{local}}} \to \ell$

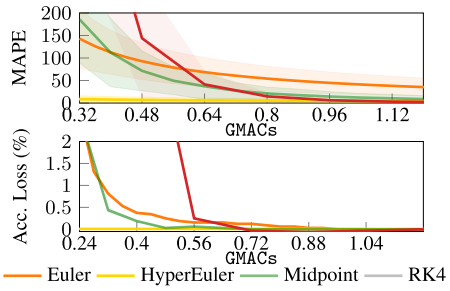

Figure 1: Test acc. loss (MNIST) and solution MAPE (avg. across batches) w.r.t `MACs`. 10^9`MACs` := 1`GMAC`. `HyperEuler` always shows higher pareto optimality. Note that 0.4 `GMACs` $\approx$ cost of 1 eval. of vector field f, 0.2 `GMACs` $\approx$ an eval. of g.

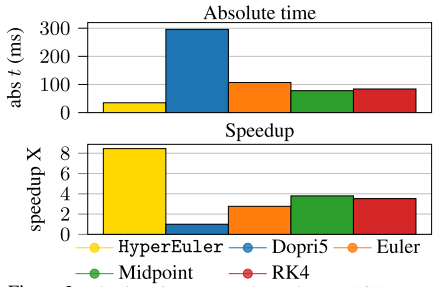

Figure 2: Absolute time (ms) and speedup, MNIST test set inference (avg. across data–batches) of various discretization schemes w.r.t the ground–truth solver `dopri5`. All fixed–step methods perform the minimum number of steps to preserve *total* test set classification accuracy loss to $< 0.1\%$.