BOLD: LEARNING OPTIMIZATION OBJECTIVES VIA BILEVEL OPTIMIZATION AND SYMBOLIC DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Many inverse problems are naturally formulated as optimization problems with stable solvers and interpretable objective functions. Recent deep learning approaches have mainly focused on learning optimizer parameters or replacing optimization pipelines with end-to-end black-box models, while the automatic discovery of optimization objectives remains comparatively underexplored. We propose BOLD, a framework for Bilevel Objective Learning and Symbolic Distillation. BOLD parameterizes optimization objectives as decomposable smooth components, trains them through finite-horizon bilevel optimization, and supports symbolic distillation of learned components into analytic expressions. We study three progressively broader instantiations: scalar-objective augmentation for finite-budget sparse recovery, learned local regularization for spring-system structure recovery, and full objective discovery for graph learning from smooth signals. Across these three settings, BOLD demonstrates complementary benefits of learning the objective itself: LP-ALISTA substantially improves finite-budget recovery over its ALISTA backbone; the learned spring-system regularizer can be distilled into an analytic penalty while largely preserving the neural teacher's support-recovery performance; and the learned graph objectives match or outperform classical Kalofolias-style baselines in most representative settings while revealing interpretable signal-distance and degree-shaping structure. Together, these results show that objective learning can improve downstream recovery while preserving an explicit optimization formulation and enabling symbolic interpretation.

1 INTRODUCTION

Many inverse problems estimate unknown variables or structures by minimizing an objective that combines data fidelity with structural regularization, making the underlying modeling assumptions and the role of each objective term explicit (Boyd & Vandenberghe, 2004; Beck & Teboulle, 2009). This paradigm underlies sparse regression, physical interaction recovery, and graph learning from smooth signals (Tibshirani, 1996; Brunton et al., 2016; Dong et al., 2016; Kalofolias, 2016; Wang et al., 2016). However, their objectives are usually hand-designed and may be suboptimal under limited computation budgets or mismatched to the statistics of a particular data distribution. Recent learning-based methods have improved these pipelines by learning optimizer parameters, unrolling iterative solvers, or replacing the optimization procedure with end-to-end neural predictors (Andrychowicz et al., 2016; Monga et al., 2021; Gregor & LeCun, 2010; Chen et al., 2018; Liu et al., 2019; Kulkarni et al., 2016; Kipf et al., 2018; Wang & Pang, 2024). These approaches often improve empirical performance, but they either keep the objective fixed or discard the objective-driven formulation altogether. This leaves open a central question: can we learn optimization objective components from data while retaining the stability and interpretability that make optimization-based methods useful?

Learning objective functions is challenging for several reasons. First, unlike an end-to-end predictor that directly outputs the target signal or structure, a learned objective influences the prediction only through the trajectory produced by an inner optimizer. Training must therefore propagate task-level supervision through iterative optimization, which can involve long unrolled trajectories, unstable gradients, and support changes induced by thresholding or projection operations (Domke, 2012; Franceschi et al., 2018; Blondel et al., 2022; Bertrand et al., 2022; Bolte et al., 2023). Second, a fully

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

black-box high-dimensional objective is difficult to interpret or symbolically regress into a useful formula (Schmidt & Lipson, 2009; Cranmer et al., 2020; La Cava et al., 2021). Third, symbolic expressions that fit learned components pointwise may fail when inserted back into an optimizer, because downstream recovery depends on gradients and rollout behavior. We propose **BOLD**, a framework for **Bilevel Objective Learning and Symbolic Distillation**. BOLD parameterizes objectives as shared decomposable smooth scalar components, learns them from task-level supervision through finite-horizon bilevel optimization, and supports their distillation into analytic expressions using symbolic regression.

These challenges motivate three design choices in BOLD. First, to learn through the optimizer rather than predict the target directly, BOLD trains explicit scalar objective components through finite-horizon solver rollouts; for long-horizon settings, a detached burn-in can shorten the differentiated tail while preserving the forward optimization trajectory. Second, to make learned objectives amenable to interpretation and symbolic distillation, BOLD decomposes them into shared low-dimensional scalar components, providing a structured trade-off between neural expressivity and symbolic distillability. Third, because pointwise symbolic fidelity need not preserve optimization behavior, BOLD reinserts symbolic candidates into the solver and selects them by validation rollout performance. Together, these choices yield a solver-aware pipeline for learning executable optimization objectives from downstream task supervision.

Experiments instantiate this progression as sparse objective augmentation, spring-system local regularization, and full graph-objective discovery. In sparse recovery, LP-ALISTA substantially improves finite-budget recovery over its ALISTA backbone across iteration budgets and most signal dimensions. In spring-system recovery, symbolic distillation produces an analytic local penalty that largely preserves the neural teacher's support-recovery performance. In graph learning, the learned objectives match or outperform Kalofolias-style baselines in most representative settings, while their symbolic forms reveal interpretable signal-distance and degree-shaping structure. Additional analyses show that solver-level selection can be critical for obtaining useful symbolic objectives, richer contextual inputs improve neural teachers but can reduce symbolic distillability, and detached burn-in stabilizes long unrolled training in the graph setting.

2 Related Work

Optimization-based inverse problems and graph structure recovery. Classical optimization-based recovery methods combine data fidelity with structural regularization. Examples include sparse regression with LASSO-type objectives (Tibshirani, 1996), iterative sparse solvers such as ISTA and FISTA (Daubechies et al., 2004; Beck & Teboulle, 2009), and data-driven formulations for physical or relational structure recovery (Brunton et al., 2016; Wang et al., 2016). Graph learning from smooth signals follows the same principle: given observed node signals, one learns an unknown graph by imposing signal smoothness, sparsity, and connectivity priors on the graph weights (Dong et al., 2016; Kalofolias, 2016; Ortega et al., 2018). These methods exemplify objective-driven recovery, but the objective terms themselves are specified a priori. BOLD instead treats part or all of the objective as the learned object, learning decomposable scalar components from supervised recovery tasks and supporting their distillation into analytic forms.

Unrolled sparse recovery and learned optimizers. LISTA introduced the idea of learning sparse recovery algorithms by unrolling ISTA-like iterations (Gregor & LeCun, 2010). Subsequent methods improve finite-step sparse recovery by learning update matrices, thresholds, stepsizes, support-selection rules, or adaptive nonlinearities, including ALISTA, NA-ALISTA, GLISTA, and related variants (Chen et al., 2018; Liu et al., 2019; Behrens et al., 2021; Wu et al., 2020; Monga et al., 2021). Methods in this line keep the sparse-coding objective fixed and improve finite-step recovery by learning how that objective is optimized. BOLD changes a different part of the pipeline: under the same finite-budget perspective, it learns an additional scalar objective component whose exact gradient augments the ALISTA update while retaining the unfolded solver structure.

Learning regularizers and variational objectives. A large body of work studies learned regularizers, learned proximal operators, and trainable variational models for inverse problems, especially in imaging (Roth & Black, 2005; Chen & Pock, 2017; Kobler et al., 2017; Hammernik et al., 2018; Fang et al., 2024; Zhang & Giannakis, 2024). Bilevel optimization has also been used to tune regularization

parameters or learn data-driven priors (Kunisch & Pock, 2013; Franceschi et al., 2018). More broadly, recent differentiable and bilevel optimization methods provide tools for learning through optimization procedures (Blondel et al., 2022; Liu et al., 2022; Petrulionyte et al., 2024). These works establish that regularization structure can be learned from data, but learned priors often remain neural penalties, proximal mappings, or task-specific modules. BOLD instead emphasizes explicit decomposable scalar objective components whose gradients can be used directly by optimization solvers and whose low-dimensional structure supports symbolic distillation into executable analytic expressions.

Symbolic regression and symbolic distillation. Symbolic regression has long been used for equation discovery and scientific law extraction (Schmidt & Lipson, 2009; Brunton et al., 2016; Udrescu & Tegmark, 2020; La Cava et al., 2021). More recent neural approaches improve symbolic search and model distillation with learned representations or transformers (Cranmer et al., 2020; Biggio et al., 2021; Kamienny et al., 2022; Shojaee et al., 2023). Unlike standard symbolic regression, which typically fits input–output relations or physical equations directly, BOLD distills expressions that define optimization objectives. We therefore select symbolic candidates by validation rollout performance rather than pointwise loss, so that executability and downstream recovery behavior are evaluated directly in the solver loop.

3 METHOD

3.1 OVERVIEW OF BOLD

We propose **BOLD**, a framework for **Bilevel Objective Learning and Symbolic Distillation**. Given an observation y and optimization variable x, BOLD represents the objective as a sum of shared scalar components:

$$E_\theta(x; y) = E_{\text{base}}(x; y) + \sum_{h \in \mathcal{H}} \sum_{\xi \in \mathcal{I}_h} \psi_{\theta_h}^{[h]} \left(u_\xi^{[h]}(x, y) \right), \tag{1}$$

where E_{base} contains any known objective terms. Each $h \in \mathcal{H}$ denotes a type of learned component, $\mathcal{I}_h$ indexes the entities of that type, and $\psi_{\theta_h}^{[h]}$ is a smooth scalar function shared across those entities and applied to their local features $u_\xi^{[h]}(x, y)$. For example, in our graph-learning instantiation, the optimization variable is the edge-weight vector w, $\mathcal{H}$ contains an edge component and a node component, and we set $E_{\text{base}} = 0$ because the goal here is to learn the full decomposable objective rather than augment a prescribed analytic one. For the edge component, $\mathcal{I}_h = \mathcal{E}$, each entity $\xi = (i, j)$ is an edge, $u_\xi^{[h]}$ contains its current weight and signal-distance features, and the same scalar function is shared over all edges; the node component is defined analogously over nodes using degree-based features. The parameters of all shared component functions are collected in $\theta = \{\theta_h\}_{h \in \mathcal{H}}$. Across the three instantiations, BOLD always learns explicit scalar objective components, but their role differs. In sparse recovery, the learned potential provides an objective-derived gradient augmentation to a retained ALISTA backbone; in the spring and graph settings, the learned components directly define the regularizer or the full objective optimized by the inner solver.

Decomposability serves as a structural inductive bias: sharing low-dimensional scalar functions across coordinates, edges, or nodes reduces objective learning to repeated local mappings, keeps the learned objective explicitly factorized, and provides a natural interface for symbolic distillation. The corresponding feature maps may still include shared or global context, so this decomposition constrains how the objective is parameterized without requiring the underlying entities to be independent.

Conceptually, BOLD can be written as a standard bilevel objective-learning problem:

$$\begin{aligned} \min_\theta \quad & \mathbb{E}_{(y, t^{\text{gt}})} \left[\mathcal{L} \left(x_\theta^\star(y), t^{\text{gt}} \right) \right] \\ \text{subject to} \quad & x_\theta^\star(y) \in \arg\min_x E_\theta(x; y), \end{aligned} \tag{2}$$

where t^{gt} is the supervised target and $\mathcal{L}$ is the task loss. This formulation makes explicit that supervision acts on the solution induced by the learned objective rather than directly on the objective values themselves.

In practice, we evaluate a learned objective through the trajectory of the instantiated solver rather than assuming an exact lower-level minimizer. The prescribed horizon may represent either a tight

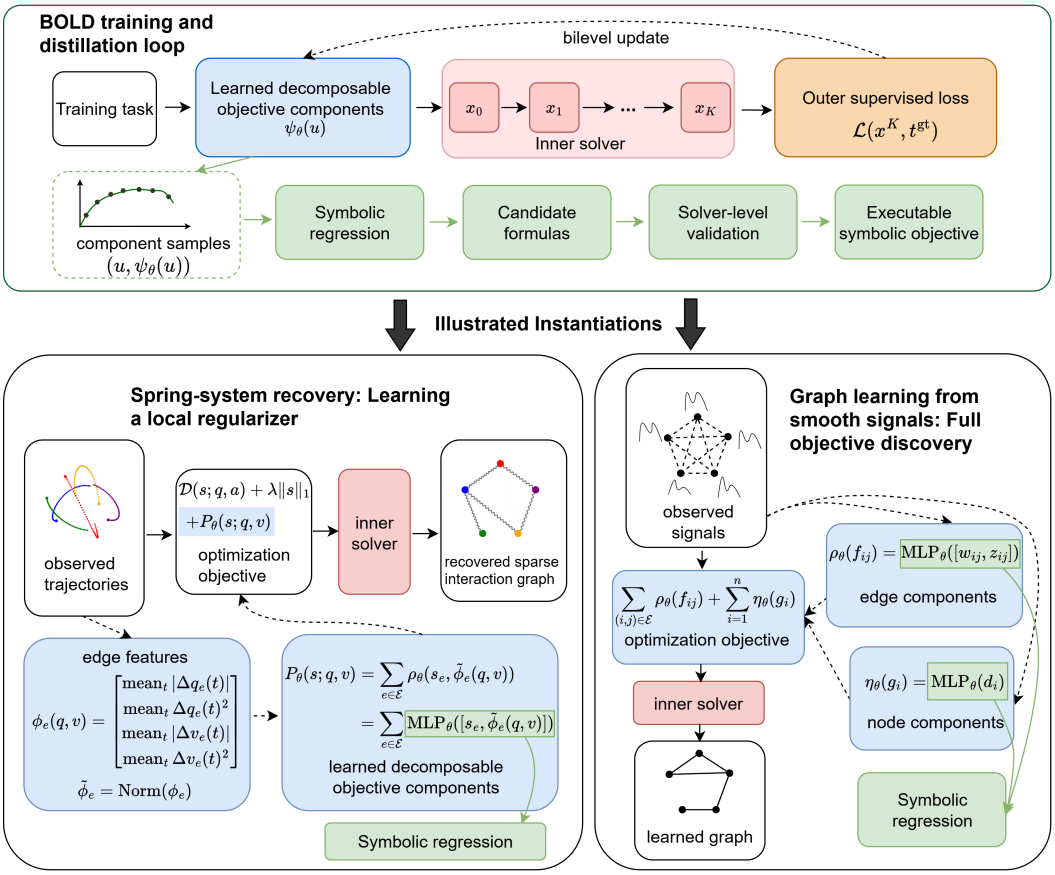

Figure 1: Overview of BOLD. The top panel shows the common bilevel training and optional symbolic-distillation loop. The bottom panels illustrate two representative instantiations: learning a local regularizer for spring-system recovery (left) and full objective discovery for graph learning from smooth signals (right). Sparse-recovery objective augmentation in Section 3.2 is omitted from the schematic for space. Colors distinguish different modules.

finite-budget regime or a longer optimization rollout, depending on the task. We therefore use the finite-horizon formulation

$$\begin{aligned} \min_{\theta} \quad & \mathbb{E}_{(y,t^{\text{gt}})}\left[\mathcal{L}\left(x_\theta^K(y), t^{\text{gt}}\right)\right] \\ \text{subject to} \quad & x_\theta^0 = x_{\text{init}}(y), \\ & x_\theta^{k+1} = \mathcal{T}_k\left(x_\theta^k; y, E_\theta\right), \qquad k = 0, \ldots, K-1, \end{aligned} \tag{3}$$

where x_{init} initializes the optimization variable, $\mathcal{T}_k$ is one step of the instantiated solver, and K is the prescribed horizon. The finite solver trajectory is therefore part of the training objective itself, and we differentiate through its retained computational graph. For long horizons, an initial burn-in segment can be detached to shorten the gradient path without changing the forward K-step trajectory. Implicit differentiation provides an alternative when the stationary lower-level solution is the quantity of interest, but is not used here (Bertrand et al., 2022; Blondel et al., 2022).

Algorithm 1 summarizes the resulting training and optional symbolic-distillation pipeline. The three instantiations below retain this common template while progressively expanding the learned objective from sparse-recovery augmentation, to a local spring regularizer, to the full graph objective.

Algorithm 1 BOLD: objective learning and optional symbolic distillation

Require: Training and validation data; component families; solver $\mathcal{T}_k$; horizon K; burn-in length $0 \le B < K$ (default $B = 0$)

Initialize objective parameters θ
for each training batch $\mathcal{B}$ **do**
  $x^0(y) \leftarrow x_{\text{init}}(y)$ for each $(y, t^{\text{gt}}) \in \mathcal{B}$
  for $k = 0, \dots, K-1$ **do**
    $x^{k+1} \leftarrow \mathcal{T}_k(x^k; y, E_\theta)$
    if $k < B$ **then**
      $x^{k+1} \leftarrow \text{stopgrad}(x^{k+1})$
    end if
  end for
  $\ell \leftarrow |\mathcal{B}|^{-1} \sum_{(y,t^{\text{gt}})\in\mathcal{B}} \mathcal{L}(x^K(y), t^{\text{gt}})$
  Update θ through the retained trajectory
  At validation intervals, update the best checkpoint by the validation criterion
end for
Freeze the validation-selected model $E_{\theta^\star}$
if symbolic distillation is enabled **then**
  Collect component inputs and distillation targets
  Generate symbolic candidates
  Reinsert candidates into the corresponding solver
  Select $\widehat{E}$ by validation rollout performance
end if
return $E_{\theta^\star}$ and $\widehat{E}$ if distilled

3.2 Sparse Recovery: Finite-Budget Objective Augmentation

For sparse recovery with $y = \Phi x^{\text{gt}} + \epsilon$, we take the canonical LASSO formulation as the reference model,

$$E_{\text{LASSO}}(x; y) = \frac{1}{2}\|y - \Phi x\|_2^2 + \lambda \|x\|_1. \tag{4}$$

ALISTA retains the corresponding ISTA-like shrinkage structure with a fixed analysis matrix W and learned layerwise steps and thresholds (Liu et al., 2019). Define

$$b^k = \Phi x^k - y, \qquad c^k = W^\top b^k. \tag{5}$$

BOLD learns a smooth scalar objective component $P_\theta(x; y)$, supplying its exact gradient $g_\theta(x; y) = \nabla_x P_\theta(x; y)$ to the finite-budget solver:

$$x^{k+1} = \mathcal{S}_{\tau_k, p_k}\left(x^k - \gamma_k c^k - \beta \nabla_x P_\theta(x^k; y)\right). \tag{6}$$

Here γ_k, τ_k retain the ALISTA parameterization, and $\mathcal{S}_{\tau_k,p_k}$ retains its support-selection shrinkage rule. One learnable scalar β, independent of γ_k, and the same potential are shared across all K iterations. We use a structured state-aware scalar parameterization, detailed in Appendix A.1, and train it through the finite solver rollout. This instantiates BOLD as objective-derived augmentation: the learned object is a shared scalar potential whose exact gradient enters every iteration, while the remaining solver mechanics are inherited from ALISTA to preserve a controlled comparison with the backbone.

3.3 Spring Systems: Learning a Local Regularizer

The second instantiation retains the finite-horizon recovery setting but asks a different question: can a learned objective component also be organized as a local regularizer and distilled into an executable analytic form? To study this in a physically grounded situation, we consider spring-system structure recovery, where trajectories of positions q, velocities v, and accelerations a are observed, and the goal is to recover nonnegative sparse edge weights $s \in \mathbb{R}_+^{|\mathcal{E}|}$ over a fixed candidate edge set. Let B be the incidence matrix of the candidate graph and let m be the common particle mass. The predicted acceleration under edge weights s is

$$\hat{a}_s(q_t) = -\frac{1}{m} B \,\text{diag}(s) B^\top q_t. \tag{7}$$

The baseline sparse recovery objective is

$$\min_{s\geq 0}\ D(s;q,a)+\lambda\|s\|_1, \qquad D(s;q,a)=\mathrm{mean}_{t,i}\left(\hat{a}_s(q_t)_i-a_{t,i}\right)^2. \tag{8}$$

As in the sparse-recovery instantiation above, training evaluates recovery after a prescribed number of inner-solver steps rather than assuming exact convergence.

BOLD augments the smooth part of this objective with an edge-wise learned penalty:

$$\min_{s\geq 0}\ D(s;q,a)+P_\theta(s;q,v)+\lambda\|s\|_1, \qquad P_\theta(s;q,v)=\sum_{e\in\mathcal{E}}\rho_\theta\Big(s_e,\tilde{\phi}_e(q,v)\Big), \tag{9}$$

where $\tilde{\phi}_e(q,v)$ is a normalized edge feature vector computed from relative displacement and velocity. The finite-horizon inner solver applies ISTA to the smooth term $D+P_\theta$ together with the nonnegative ℓ_1 proximal step. The learned local regularizer is then distilled into an analytic form and selected by downstream solver performance. Implementation and symbolic-distillation details are given in Appendix A.2.

3.4 Graph Learning: Full Objective Discovery

The third instantiation changes both the scope and the role of objective learning. Whereas the preceding settings augment a known recovery objective, here BOLD learns the entire decomposable objective itself. The goal is therefore not primarily to improve recovery under a tight iteration budget, but to obtain a directly optimizable objective that remains effective over long solver rollouts.

We consider graph learning from smooth signals on N_g nodes, with nonnegative edge weights $w\in\mathbb{R}_+^{|\mathcal{E}|}$. For observed node signals s_i, let $z_{ij}=\|s_i-s_j\|_2^2$ denote the pairwise signal distance and $d_i=\sum_{j\neq i}w_{ij}$ the induced node degree. Classical Kalofolias-style objectives combine signal-distance, edge-weight, and degree regularization (Kalofolias, 2016). We instead set $E_{\mathrm{base}}=0$ and learn shared edge and node components:

$$E_\theta(w;z)=\sum_{(i,j)\in\mathcal{E}}\rho_\theta(f_{ij}(w,z))+\sum_{i=1}^{N_g}\eta_\theta(g_i(w,z)). \tag{10}$$

In the low-prior setting used in the main experiments, $f_{ij}=(w_{ij},z_{ij})$ and $g_i=(d_i)$, so the model receives primitive edge weights, signal distances, and degrees rather than an explicit analytic graph objective. The learned objective is optimized by projected gradient descent with step size η_{step},

$$w^{k+1}=\Pi_{\mathbb{R}_+^{|\mathcal{E}|}}\left(w^k-\eta_{\mathrm{step}}\nabla_w E_\theta(w^k;z)\right). \tag{11}$$

Training uses the common finite-horizon bilevel formulation above, with the prescribed horizon serving as a long optimization rollout rather than a tight computation budget. The task loss is the weight-recovery MSE between w^K and w^{gt}. Implementation details are given in Appendix A.3.

3.5 Symbolic Distillation and Solver-Level Selection

For the spring and graph instantiations, symbolic distillation forms a second stage after neural objective learning. Symbolic regression generates analytic candidates from the learned local components or their solver-relevant targets. Because pointwise approximation quality does not necessarily preserve optimization behavior, we reinsert each candidate into the corresponding solver and select the symbolic representation by validation rollout performance. The selected representation is then evaluated on held-out tasks as an executable regularizer or objective. Detailed export, regression, and selection procedures are given in Appendix A.4.

4 Experiments

We evaluate BOLD across three progressively broader scopes of objective learning. The experiments are designed to answer three questions: (1) can a learned scalar objective component improve sparse recovery under limited iteration budgets? (2) can a learned local regularizer be distilled into an executable symbolic form in a physically meaningful structure recovery task? (3) can BOLD learn and symbolically distill the full objective for graph learning from smooth signals?

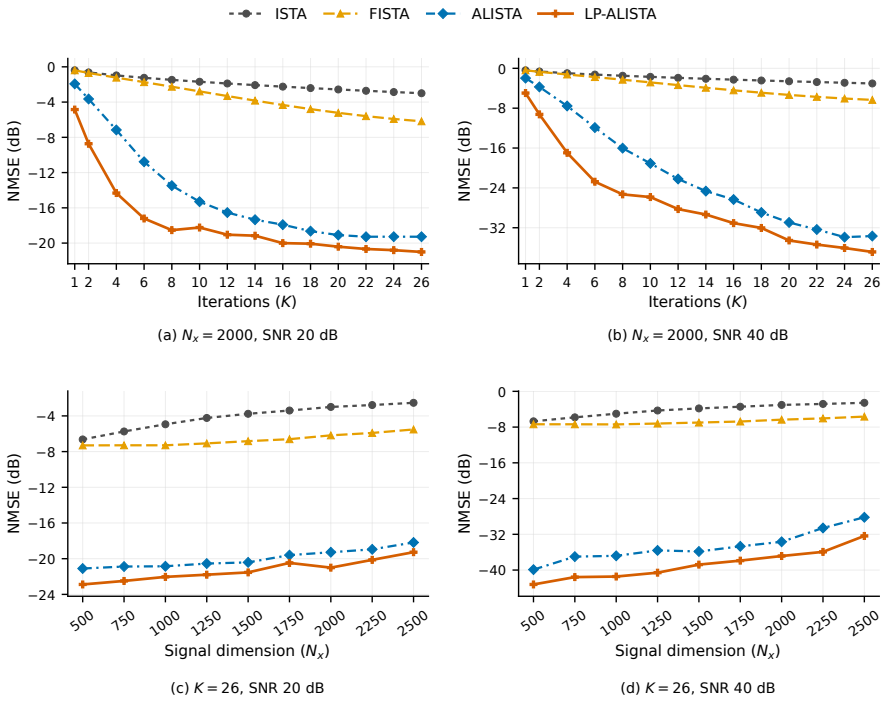

Figure 2: Finite-budget sparse regression on the independent held-out test set. Top: varying K at $N_x = 2000$; bottom: varying N_x at $K = 26$. Lower NMSE is better.

4.1 Finite-Budget Sparse Regression

Setup. We study synthetic sparse reconstruction from $y = \Phi x^{\mathrm{gt}} + \epsilon$, with column-normalized Gaussian Φ, measurement dimension $M = 250$, expected sparsity $S = 50$, and SNR 20 or 40 dB. We compare ISTA, FISTA, ALISTA, and LP-ALISTA across iteration budgets K at $N_x = 2000$ and across signal dimensions N_x at $K = 26$. LP-ALISTA is trained separately for each setting and selected by validation NMSE; all reported results use the same independent held-out test set.

Results. Figure 2 shows that LP-ALISTA improves finite-budget recovery over its ALISTA backbone across both sweeps and noise levels, with the largest gains at small iteration budgets. This supports scalar-objective augmentation as an effective way to improve recovery without replacing the underlying unfolded solver.

4.2 Spring-System Structure Recovery and Symbolic Penalty Distillation

We evaluate BOLD on spring-system structure recovery, where the goal is to infer sparse pairwise interactions from observed trajectories. Table 1 reports results for $N_p \in \{6, 20\}$ and $K \in \{30, 100\}$, with SR-LP-ISTA denoting the solver using the distilled symbolic regularizer.

LP-ISTA substantially improves support recovery over ISTA, LISTA, and ALISTA. Its symbolic counterpart remains close to the neural teacher for $N_p = 6$ and further improves F1 and support MSE in the harder $N_p = 20$ settings, showing that the learned local regularizer can be distilled without sacrificing its solver-level effectiveness.

Symbolic penalty interpretation. The selected symbolic penalties repeatedly couple the current edge weight with trajectory-dependent features. In particular, we observe terms involving $s_e - \tilde{a}_e$, where $\tilde{a}_e$ is the normalized version of $\mathrm{mean}_t\, |\Delta q_e(t)|$. This suggests that the learned penalty does not simply impose a uniform sparsity bias. Instead, it adjusts the marginal cost of an edge according to whether the current weight estimate is compatible with the observed relative motion of the

Table 1: Results on spring-system structure recovery with metrics: Precision↑, Recall↑, F1↑, Exact Support↑, Weight MSE↓, and Support MSE↓. Exact Support measures the fraction of test examples for which the recovered support exactly matches the ground-truth support. Best results within each (N_p, K) block are boldfaced.

Model	N_p	K	Metrics					
			Precision	Recall	F1	Exact Support	Weight MSE	Support MSE
ISTA	6	30	0.5875	**0.9755**	0.7010	0.2070	0.0554	0.1479
LISTA			0.6168	0.9564	0.7174	0.2500	0.0563	0.1266
ALISTA			0.5945	0.9611	0.7016	0.2305	**0.0542**	0.1383
LP-ISTA			**0.9294**	0.9406	**0.9248**	**0.7265**	0.0589	**0.0302**
SR-LP-ISTA			0.9219	0.9419	0.9203	**0.7265**	0.0655	0.0323
ISTA	6	100	0.8021	0.9836	0.8617	0.5510	0.0260	0.0630
LISTA			0.7962	0.9733	0.8539	0.5470	0.0272	0.0646
ALISTA			0.7957	0.9733	0.8540	0.5510	0.0267	0.0648
LP-ISTA			0.9538	**0.9846**	**0.9641**	**0.8670**	**0.0215**	**0.0159**
SR-LP-ISTA			**0.9612**	0.9391	0.9454	0.8480	0.0257	0.0167
ISTA	20	30	0.2818	**0.8827**	0.4260	0.0000	**0.1457**	0.3538
LISTA			0.2963	0.8587	0.4369	0.0000	0.1472	0.3288
ALISTA			0.2967	0.8582	0.4381	0.0000	0.1473	0.3264
LP-ISTA			0.4214	0.7068	0.5248	0.0000	0.1516	0.1934
SR-LP-ISTA			**0.5045**	0.6042	**0.5410**	0.0000	0.1461	**0.1525**
ISTA	20	100	0.3369	0.8786	0.4856	0.0000	0.1206	0.2781
LISTA			0.3375	**0.8906**	0.4875	0.0000	0.1177	0.2781
ALISTA			0.3576	0.8510	0.4997	0.0000	0.1231	0.2534
LP-ISTA			0.5302	0.7926	0.6303	0.0000	0.1157	0.1420
SR-LP-ISTA			**0.6421**	0.6866	**0.6605**	0.0000	**0.1152**	**0.1065**

corresponding particle pair. A representative quadratic marginal has the simplified form

$$\widehat{g}_e(s) = \kappa_e(s_e - A_e)(s_e - B_e) + C_e,$$

where A_e, B_e, and C_e depend on local trajectory features. Its sign and magnitude therefore vary with both the current edge weight and the observed motion; integrating this representative marginal yields a feature-dependent cubic local penalty. Full solver-selected symbolic expressions are provided in Appendix C.1.

4.3 Graph Learning from Smooth Signals

We evaluate full objective discovery on graph learning from smooth signals, comparing the classical Kalofolias baseline with the learned neural objective BOLD and its distilled symbolic counterpart SR-BOLD.

Representative settings. We consider RGG+Tikhonov, RGG+Heat, BA+Tikhonov, and BA+Heat for $N_g \in \{20, 100\}$. Table 2 shows that BOLD matches or outperforms Kalofolias in most representative settings, with particularly strong gains on BA graphs. SR-BOLD generally preserves support recovery and can further improve F1, although some weighted-error metrics degrade after distillation.

Representative symbolic graph objective. Across the selected symbolic objectives, the edge and node components often admit the qualitative forms

$$\widehat{\rho}(w, z) \approx a(z)w + b(z)w^2, \qquad \widehat{\eta}(d) \approx (d - d_0)R(d) \text{ or } R(d, \log d),$$

where a, b, and R are learned symbolic functions. These expressions recover Kalofolias-style mechanisms—signal-distance edge costs, edge-weight shaping, and degree-dependent density control— without reproducing the hand-designed objective exactly. Full solver-selected expressions are given in Appendix C.2.

Table 2: Results on representative graph learning from smooth signals settings, best results in each setting-metric group are boldfaced.

N_g	Setting	Method	F1 ↑	edge ℓ_1 ↓	edge ℓ_2 ↓	degree ℓ_1 ↓	degree ℓ_2 ↓
20	RGG + Tikhonov	Kalofolias	**0.958**	**0.245**	0.288	0.156	0.167
		BOLD	0.935	0.256	**0.280**	**0.141**	**0.156**
		SR-BOLD	0.940	0.267	0.300	0.176	0.189
	RGG + Heat	Kalofolias	0.874	**0.395**	**0.415**	**0.225**	**0.244**
		BOLD	**0.889**	0.400	0.425	0.252	0.271
		SR-BOLD	0.869	0.552	0.548	0.402	0.432
	BA + Tikhonov	Kalofolias	0.953	0.954	0.577	0.205	0.250
		BOLD	0.948	**0.265**	**0.304**	**0.077**	**0.089**
		SR-BOLD	**0.956**	0.463	0.398	0.089	0.094
	BA + Heat	Kalofolias	0.844	0.824	0.593	0.233	0.296
		BOLD	0.867	**0.486**	**0.468**	**0.131**	**0.168**
		SR-BOLD	**0.878**	0.557	0.560	0.199	0.233
100	RGG + Tikhonov	Kalofolias	0.920	0.270	0.320	**0.043**	**0.051**
		BOLD	**0.933**	**0.244**	**0.295**	0.098	0.118
		SR-BOLD	0.922	0.340	0.354	0.134	0.185
	RGG + Heat	Kalofolias	0.858	**0.406**	0.458	**0.121**	**0.144**
		BOLD	**0.864**	0.439	0.438	0.148	0.174
		SR-BOLD	0.853	0.434	**0.431**	0.153	0.184
	BA + Tikhonov	Kalofolias	0.922	1.490	0.763	0.205	0.315
		BOLD	0.965	**0.217**	**0.259**	**0.084**	**0.118**
		SR-BOLD	**0.986**	0.229	0.288	0.089	0.141
	BA + Heat	Kalofolias	0.861	0.874	0.607	0.147	0.294
		BOLD	0.904	**0.411**	**0.408**	**0.107**	**0.153**
		SR-BOLD	**0.916**	0.512	0.480	0.127	0.165

4.4 Extended Experiments

Appendix B provides several additional insights into BOLD's design. Solver-level selection can be crucial: symbolic expressions with good pointwise fit can perform poorly once reinserted into the optimizer, whereas rollout-based selection closely preserves the neural teacher. Feature ablations further reveal an expressivity–distillability trade-off, and detached burn-in stabilizes long unrolled training while retaining strong recovery performance. Long-horizon evaluations further show that both BOLD and its symbolic counterpart remain effective under substantially extended optimization, while direct transfer reveals graph-family shifts as a challenge shared with the classical baseline. A jointly trained all-setting objective provides a promising route toward greater cross-setting robustness.

5 Conclusion and Limitations

We presented BOLD, a bilevel framework for learning decomposable optimization-objective components from task-level supervision and distilling them into executable analytic forms. Across three progressively broader instantiations, BOLD augments the ALISTA backbone with a learned scalar potential, learns and distills a local spring-system regularizer, and discovers a full decomposable graph-learning objective. These results show that explicit objective components can improve recovery while preserving structured optimization and enabling symbolic interpretation.

BOLD has several limitations. First, decomposable coordinate-, edge-, or node-level factors facilitate sharing and symbolic distillation but do not capture arbitrary global or higher-order objectives. Second, bilevel training and neural objective parameterization complicate theoretical analysis, so we do not provide general convergence or recovery guarantees. Third, learned objectives can remain distribution-dependent, while greater neural expressivity can reduce symbolic distillability. Finally, our experiments are predominantly synthetic; extension to real inverse problems remains future work.

AI USE STATEMENT

Generative AI tools were used to provide feedback on experimental and methodological design, assist with method implementation and debugging, support diagnosis and interpretation of experimental results, and assist with code and manuscript editing. All AI-assisted work was reviewed by the authors.

REPRODUCIBILITY STATEMENT

Section 3 and Algorithm 1 specify the common BOLD training and symbolic-distillation procedure. Appendix A provides the implementation details for sparse recovery, spring-system recovery, and graph learning, together with symbolic-regression procedures, experimental hyperparameters, dataset splits, and compute resources. Appendix B reports the main ablations, stability analyses, long-horizon evaluations, and cross-setting transfer experiments, while Appendix C provides the selected symbolic expressions. We additionally provide anonymous supplementary source code and experiment configurations to support reproduction of the reported results.

REFERENCES

Marcin Andrychowicz, Misha Denil, Sergio Gómez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *Advances in Neural Information Processing Systems*, 29:3981–3989, 2016.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

Freya Behrens, Jonathan Sauder, and Peter Jung. Neurally augmented ALISTA. In *International Conference on Learning Representations*, 2021.

Quentin Bertrand, Quentin Klopfenstein, Mathurin Massias, Mathieu Blondel, Samuel Vaiter, Alexandre Gramfort, and Joseph Salmon. Implicit differentiation for fast hyperparameter selection in non-smooth convex learning. *Journal of Machine Learning Research*, 23(149):1–43, 2022.

Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In *International Conference on Machine Learning*, pp. 936–945. PMLR, 2021.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *Advances in Neural Information Processing Systems*, 35:5230–5242, 2022.

Jérôme Bolte, Edouard Pauwels, and Samuel Vaiter. One-step differentiation of iterative algorithms. *Advances in Neural Information Processing Systems*, 36:77089–77103, 2023.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

Xiaohan Chen, Jialin Liu, Zhangyang Wang, and Wotao Yin. Theoretical linear convergence of unfolded ISTA and its practical weights and thresholds. *Advances in Neural Information Processing Systems*, 31:9079–9089, 2018.

Yunjin Chen and Thomas Pock. Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1256–1272, 2017.

Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems*, 33:17429–17442, 2020.

Ingrid Daubechies, Michel Defrise, and Christine de Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57(11):1413–1457, 2004.

Justin Domke. Generic methods for optimization-based modeling. In *International Conference on Artificial Intelligence and Statistics*, pp. 318–326. PMLR, 2012.

Xiaowen Dong, Dorina Thanou, Pascal Frossard, and Pierre Vandergheynst. Learning laplacian matrix in smooth graph signal representations. *IEEE Transactions on Signal Processing*, 64(23): 6160–6173, 2016.

Zhenghan Fang, Sam Buchanan, and Jeremias Sulam. What's in a prior? learned proximal networks for inverse problems. In *International Conference on Learning Representations*, 2024.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pp. 1568–1577. PMLR, 2018.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *International Conference on Machine Learning*, pp. 399–406, 2010.

Kerstin Hammernik, Teresa Klatzer, Erich Kobler, Michael P. Recht, Daniel K. Sodickson, Thomas Pock, and Florian Knoll. Learning a variational network for reconstruction of accelerated MRI data. *Magnetic Resonance in Medicine*, 79(6):3055–3071, 2018.

Vassilis Kalofolias. How to learn a graph from smooth signals. In *International Conference on Artificial Intelligence and Statistics*, pp. 920–929. PMLR, 2016.

Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. *Advances in Neural Information Processing Systems*, 35:10269–10281, 2022.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pp. 2688–2697. PMLR, 2018.

Erich Kobler, Teresa Klatzer, Kerstin Hammernik, and Thomas Pock. Variational networks: Connecting variational methods and deep learning. In *German Conference on Pattern Recognition*, pp. 281–293. Springer, 2017.

Kuldeep Kulkarni, Suhas Lohit, Pavan Turaga, Ronan Kerviche, and Amit Ashok. ReconNet: Non-iterative reconstruction of images from compressively sensed measurements. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 449–458, 2016.

Karl Kunisch and Thomas Pock. A bilevel optimization approach for parameter learning in variational models. *SIAM Journal on Imaging Sciences*, 6(2):938–983, 2013.

William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance. In *Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

Bo Liu, Mao Ye, Stephen Wright, Peter Stone, and Qiang Liu. BOME! bilevel optimization made easy: A simple first-order approach. *Advances in Neural Information Processing Systems*, 35: 17248–17262, 2022.

Jialin Liu, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. ALISTA: Analytic weights are as good as learned weights in LISTA. In *International Conference on Learning Representations*, 2019.

Vishal Monga, Yuelong Li, and Yonina C. Eldar. Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing. *IEEE Signal Processing Magazine*, 38(2):18–44, 2021.

Antonio Ortega, Pascal Frossard, Jelena Kovačević, José M. F. Moura, and Pierre Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106 (5):808–828, 2018.

Ieva Petrulionyte, Julien Mairal, and Michael Arbel. Functional bilevel optimization for machine learning. *Advances in Neural Information Processing Systems*, 37:14016–14065, 2024.

Stefan Roth and Michael J. Black. Fields of experts: A framework for learning image priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pp. 860–867, 2005.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan K. Reddy. Transformer-based planning for symbolic regression. *Advances in Neural Information Processing Systems*, 36: 45907–45919, 2023.

Robert Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

Silviu-Marian Udrescu and Max Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

Aoran Wang and Jun Pang. Structural inference with dynamics encoding and partial correlation coefficients. In *International Conference on Learning Representations*, 2024.

Wen-Xu Wang, Ying-Cheng Lai, and Celso Grebogi. Data based identification and prediction of nonlinear and complex dynamical systems. *Physics Reports*, 644:1–76, 2016.

Kailun Wu, Yiwen Guo, Ziang Li, and Changshui Zhang. Sparse coding with gated learned ISTA. In *International Conference on Learning Representations*, 2020.

Yilang Zhang and Georgios B. Giannakis. Meta-learning priors using unrolled proximal networks. In *International Conference on Learning Representations*, 2024.

A Implementation Details

A.1 Learned Scalar Objective Component for Sparse Recovery

Unlike the spring and graph instantiations, the sparse-recovery setting does not replace the complete inner objective optimized by the solver. Instead, we deliberately retain the ALISTA backbone and test whether an explicit learned scalar potential can improve its finite-budget trajectory through an exact gradient contribution. The layerwise ALISTA steps, thresholds, and support-selection schedule are therefore kept to align the learned variant with its ALISTA baseline, rather than being interpreted as components of a single learned total objective. LP-ALISTA retains the original ALISTA backbone and supplies it with the exact gradient of one shared scalar potential. The potential combines a structured neural scalar with two analytic scalar terms having learned coefficients.

Backbone quantities and state features. For $y \in \mathbb{R}^M$, $x \in \mathbb{R}^{N_x}$, and fixed matrices $\Phi, W \in \mathbb{R}^{M \times N_x}$, define

$$b = \Phi x - y, \qquad c = W^\top b. \tag{12}$$

We use smoothing constant $\varepsilon = 0.03$, distinct from the observation noise ϵ, and radius $R = 3$. The state-aware features are

$$\begin{aligned} r_\varepsilon &= \frac{1}{M}\sum_{j=1}^{M}\sqrt{b_j^2+\varepsilon^2}, \quad u_\varepsilon = \frac{1}{N_x}\sum_{i=1}^{N_x}\sqrt{c_i^2+\varepsilon^2}, \\ d_i &= 1+\frac{x_i^2+c_i^2}{R^2}, \qquad q_i = \left[\frac{x_i^2}{d_i}, \frac{x_i c_i}{d_i}, \frac{c_i^2}{d_i}\right], \\ v_i &= [q_i,\ s_r r_\varepsilon,\ s_u u_\varepsilon]. \end{aligned} \tag{13}$$

The rational denominator tempers the quadratic features. The fixed conditioning constants are $s_r = s_u = 20$. They were estimated once as inverse average smoothed residual magnitudes evaluated at ground-truth training signals, using 4096 training examples at SNR20 with signal/noise seeds 10001/20001. They are not learned, recomputed along the rollout, or calibrated on validation/test data. At inference, the features depend only on x, y, Φ, W, not ground truth.

Shared scalar potential. LP-ALISTA uses a coordinate-shared scalar potential $P_\theta(x; y)$. A two-layer width-64 MLP maps each state feature $v_i \in \mathbb{R}^5$ to coefficients of a local quadratic form,

$$P_\theta^{\text{neural}}(x; y) = \sum_{i=1}^{N_x} \frac{\frac{1}{2}x_i^2 A_\theta(v_i) + x_i c_i B_\theta(v_i) + \frac{1}{2}c_i^2 C_\theta(v_i)}{d_i}. \tag{14}$$

The same network is shared across coordinates and iterations.

Exact-gradient augmentation. At every inner iteration, the learned contribution is obtained by exact automatic differentiation, $g_\theta(x; y) = \nabla_x P_\theta(x; y)$, including all state-dependent features. The same potential is shared across the full rollout, and no learned vector correction is introduced. The update is

$$x^{k+1} = \mathcal{S}_{\tau_k, p_k}\left(x^k - \gamma_k c^k - \beta \nabla_x P_\theta(x^k; y)\right), \qquad k = 0, \ldots, K-1. \tag{15}$$

A single learnable scalar β is shared across layers and is independent of the ALISTA step sizes.

ALISTA backbone. We retain the original ALISTA layerwise steps, thresholds, and support-selection rule. For $p > 0$, the implementation keeps entries strictly above the p-th largest magnitude unchanged and soft-thresholds the rest, preserving the original tie behavior. For $S = 50$, the layerwise counts $(p_k)_{k=0}^{K-1}$ are the integer-converted K equally spaced values from $\lfloor 60/K \rfloor$ to 60.

Initialization and differentiation. The potential output is initialized to zero, so the initial trajectory matches the bare ALISTA backbone, while β is initialized to 0.5. Training differentiates through the inner exact-gradient calculation using higher-order automatic differentiation; evaluation uses the same local objective gradient without retaining the higher-order training graph.

A.2 Spring-System Structure Recovery Details

We consider the one-dimensional spring system used in our experiments, with N_p particles connected by pairwise springs over an unknown interaction graph. Let $q_i(t), v_i(t), a_i(t) \in \mathbb{R}$ denote the position, velocity, and acceleration of particle i at time t, and let m be the common particle mass. For an edge $e = (i, j)$ with spring weight $s_e \geq 0$, the spring force is proportional to the relative displacement between the two particles. In the linearized recovery model used here, rest-length effects are omitted, and the acceleration of particle i is written as a weighted sum of relative displacements:

$$a_i(t) = -\frac{1}{m} \sum_{j \neq i} s_{ij}\big(q_i(t) - q_j(t)\big), \tag{16}$$

Using the oriented incidence matrix $B \in \mathbb{R}^{N_p \times |\mathcal{E}|}$, the predicted acceleration for an edge-weight vector $s \in \mathbb{R}_+^{|\mathcal{E}|}$ is

$$\hat{a}_s(q_t) = -\frac{1}{m} B \operatorname{diag}(s) B^\top q_t. \tag{17}$$

The data-fidelity term used by the solver is the mean squared acceleration error

$$D(s; q, a) = \operatorname{mean}_{t,i} \left(\hat{a}_s(q_t)_i - a_{t,i}\right)^2. \tag{18}$$

The baseline sparse recovery problem is

$$\min_{s \geq 0} \; D(s; q, a) + \lambda \|s\|_1. \tag{19}$$

BOLD adds an edge-wise decomposable smooth penalty to the objective:

$$P_\theta(s; q, v) = \sum_{e \in \mathcal{E}} \rho_\theta\Big(s_e, \tilde{\phi}_e(q, v)\Big), \tag{20}$$

so that the learned objective is

$$\min_{s \geq 0} \; D(s; q, a) + P_\theta(s; q, v) + \lambda \|s\|_1. \tag{21}$$

For each edge e, the unnormalized edge features are

$$\phi_e(q, v) = \left[\text{mean}_t\, |\Delta q_e(t)|,\ \text{mean}_t\, \Delta q_e(t)^2,\ \text{mean}_t\, |\Delta v_e(t)|,\ \text{mean}_t\, \Delta v_e(t)^2\right], \tag{22}$$

where $\Delta q_e(t)$ and $\Delta v_e(t)$ denote the scalar relative displacement and velocity along edge e. A feature normalizer Norm fitted on the training set maps the raw statistics to $\tilde{\phi}_e = \text{Norm}(\phi_e)$. The local penalty is represented by a shared edge-wise MLP:

$$\rho_\theta(s_e, \tilde{\phi}_e) = \text{MLP}_\theta\left([s_e, \tilde{\phi}_e]\right). \tag{23}$$

The inner solver is ISTA for the composite objective in (21). Starting from $s^0 = 0$, each iteration applies a gradient step on the smooth term $D + P_\theta$, followed by the proximal operator of the nonnegative ℓ_1 term:

$$s^{k+1} = \text{ReLU}\left(s^k - \eta_k \nabla_s \left[D(s^k; q, a) + P_\theta(s^k; q, v)\right] - \tau_k\right), \tag{24}$$

with $\tau_k = \eta_k \lambda$ for the baseline ISTA threshold. This corresponds to nonnegative soft-thresholding. The spring-system solver introduces no separate learned scale on the penalty gradient; the learned penalty enters directly through the smooth objective gradient $\nabla_s P_\theta$. The final linear layer of the MLP in (23) is initialized to zero, making $P_\theta(s; q, v) = 0$ and $\nabla_s P_\theta(s; q, v) = 0$ at initialization. Therefore, the initial unfolded solver exactly matches the baseline ISTA solver for (19).

A.3 Graph Objective Discovery Details

This appendix provides additional implementation details for the graph-learning experiments, including graph and signal generation, feature construction, and the projected inner solver used for optimizing learned and symbolic objectives.

Candidate edge set and ground-truth graph. For a graph with N_g nodes, the candidate edge set is the complete undirected edge set

$$\mathcal{E} = \{(i, j) : 1 \le i < j \le N_g\},$$

with $N_e := |\mathcal{E}|$. The ground-truth graph is represented by nonnegative edge weights $w^{\text{gt}} \in \mathbb{R}_+^{N_e}$, and the corresponding weighted adjacency matrix is denoted by A^{gt}. We evaluate two graph families.

Random geometric graphs (RGG). In the RGG setting, nodes are sampled as points $p_i \in [0, 1]^2$. Edges are formed between nodes that are close in Euclidean space, for example by a radius rule or nearest-neighbor rule depending on the experiment configuration. Edge weights are assigned as nonnegative functions of geometric proximity, so nearby nodes tend to have stronger connections. This construction produces spatially local graphs with smooth geometric structure.

Barabási–Albert graphs (BA). In the BA setting, the graph is generated by preferential attachment. Nodes are added sequentially, and each new node connects to existing nodes with probability proportional to their current degrees. This produces graphs with heterogeneous degree distributions and hub nodes, making the degree structure substantially different from RGG graphs.

Graph Laplacian. Given the ground-truth weighted adjacency matrix A^{gt}, we form the degree matrix

$$D_{ii}^{\text{gt}} = \sum_j A_{ij}^{\text{gt}},$$

and the combinatorial Laplacian

$$L^{\text{gt}} = D^{\text{gt}} - A^{\text{gt}}.$$

The Laplacian controls signal smoothness: a signal $x \in \mathbb{R}^{N_g}$ is smooth on the graph when

$$x^\top L^{\text{gt}} x = \sum_{i<j} A_{ij}^{\text{gt}} (x_i - x_j)^2$$

is small.

Smooth signal generation. For each graph, we generate node signals by applying a graph low-pass mechanism to random input noise. We use two standard smooth-signal models.

Tikhonov smoothing. Let $\xi \in \mathbb{R}^{N_g}$ be an i.i.d. random noise vector. A Tikhonov-smoothed signal is obtained as the solution of

$$x = \arg\min_u \frac{1}{2}\|u - \xi\|_2^2 + \frac{\kappa_{\mathrm{Tik}}}{2} u^\top L^{\mathrm{gt}} u.$$

The closed-form solution is

$$x = (I + \kappa_{\mathrm{Tik}} L^{\mathrm{gt}})^{-1} \xi.$$

Thus high-frequency components with respect to the graph Laplacian are suppressed, and the smoothing strength is controlled by κ_{Tik}.

Heat diffusion. In the heat model, signals are generated by applying the graph heat kernel to random noise:

$$x = \exp(-\tau L^{\mathrm{gt}})\xi.$$

The parameter τ controls the diffusion time. Larger τ yields smoother signals by attenuating high-Laplacian-eigenvalue components more strongly. This model corresponds to the solution of the graph heat equation

$$\frac{dx(t)}{dt} = -L^{\mathrm{gt}} x(t), \qquad x(0) = \xi,$$

evaluated at time $t = \tau$.

For each generated task, multiple smooth node signals are sampled and stacked into a signal matrix. The input edge statistic used by the graph-learning objective is the squared pairwise signal distance

$$z_{ij} = \|s_i - s_j\|_2^2,$$

where s_i denotes the vector of observed signal values at node i across samples.

Feature modes. The implementation supports several edge feature modes. For an edge $e = (i, j)$, the available modes include

$$\texttt{minimal}: \quad f_{ij} = (w_{ij}, z_{ij}), \tag{25}$$

$$\texttt{basic}: \quad f_{ij} = (w_{ij}, z_{ij}, w_{ij} z_{ij}, w_{ij}^2), \tag{26}$$

$$\texttt{context_d}: \quad f_{ij} = (w_{ij}, z_{ij}, d_i + d_j, |d_i - d_j|), \tag{27}$$

$$\texttt{context_z}: \quad f_{ij} = (w_{ij}, z_{ij}, c_i + c_j, |c_i - c_j|), \tag{28}$$

$$\texttt{context_dz}: \quad f_{ij} = (w_{ij}, z_{ij}, d_i + d_j, |d_i - d_j|, c_i + c_j, |c_i - c_j|). \tag{29}$$

Here

$$d_i = \sum_{e \in \mathcal{E}:\, i \in e} w_e$$

is the current degree induced by the optimization variable w, and

$$c_i = \frac{1}{N_g - 1} \sum_{j \neq i} z_{ij}$$

is a static node context derived from the signal-distance geometry. The node component supports corresponding degree-based and context-based features, including

$$g_i = (d_i), \qquad g_i = (d_i, \log(d_i + \varepsilon_{\mathrm{deg}}), d_i^2), \qquad g_i = (d_i, c_i).$$

Here $\varepsilon_{\mathrm{deg}} > 0$ is a fixed numerical stabilizer used in the log-degree feature. In the low-prior setting emphasized in the main text, we use the minimal feature maps

$$f_{ij} = (w_{ij}, z_{ij}), \qquad g_i = (d_i),$$

so the learned objective is not directly given analytic Kalofolias-style terms such as wz, w^2, or $\log d$. Edge, degree, and context features are normalized using statistics fitted on the training set.

Projected inner solver. The inner optimization uses projected gradient descent to enforce nonnegative edge weights. Starting from a constant initialization

$$w^0 = w_{\text{init}}\mathbf{1},$$

each step is

$$w^{k+1} = \Pi_{\mathbb{R}_+^{N_e}}\left(w^k - \eta_{\text{step}}\nabla_w E(w^k; z)\right) = \max\left(w^k - \eta_{\text{step}}\nabla_w E(w^k; z), 0\right),$$

where η_{step} is the projected-gradient step size and E is either the learned neural objective or a distilled symbolic objective. During bilevel training, the implementation may use a detached burn-in followed by a differentiable tail: burn-in steps are run with projected gradient updates but detached from the autograd graph, while the final tail steps are differentiated through for updating the objective parameters.

A.4 Symbolic Regression Implementation

We use symbolic regression to convert trained neural components into analytic expressions that can be inserted back into the corresponding optimizer. The implementation follows a two-stage procedure: candidate generation by component-level fitting, followed by solver-level candidate selection. Symbolic candidate selection optimizes downstream rollout behavior and does not impose convexity or coercivity; our claims concern execution under the prescribed solver trajectories rather than existence of a global minimizer over the full feasible domain.

Component-level datasets. For each trained neural teacher, we first export a dataset of component inputs and targets. In the graph-learning experiments, the exported file contains samples for the learned edge component and node-degree component. The symbolic regression targets are the component values,

$$\rho_\theta(f_e(w, z)) \quad \text{and} \quad \eta_\theta(g_i(w, z)),$$

so that the recovered expressions define an objective directly. In the low-prior setting used in the main experiments, the symbolic feature modes are

$$f_e = (w_e, z_e), \qquad g_i = (d_i).$$

The implementation also supports richer feature modes such as products, squared weights, endpoint degree context, and signal-distance context, but these are not used in the main low-prior graph results.

For the spring-system experiments, the optimizer uses the derivative of the learned local penalty with respect to the current edge weight. We therefore distill the local marginal correction

$$\widehat{g}_e(s_e) \approx \frac{\partial \rho_\theta(s_e, \tilde{\phi}_e)}{\partial s_e},$$

where $\tilde{\phi}_e$ contains the same normalized edge-level trajectory statistics used by the neural teacher and exported for symbolic regression. The corresponding symbolic energy can be obtained by integrating $\widehat{g}_e$ with respect to s_e, up to an additive constant independent of s_e.

PySR candidate generation. Given an exported dataset $\{(u_r, y_r)\}_{r=1}^{N_{\text{SR}}}$, we fit symbolic expressions using PySR. Before fitting, invalid samples are removed and the dataset is optionally subsampled to the configured maximum number of points. Candidate expressions are generated with a fixed operator set. For graph objectives, the edge component uses unary operators $\sqrt{\cdot}$ and $\log$, while the degree component uses $\log$; both use binary operators $+, -, \times, /$. For spring-system penalty gradients, PySR is applied to the exported gradient target with the search budget reported in Table 5.

The symbolic regression script stores the full Pareto frontier of candidate expressions as a hall-of-fame table and also writes the best pointwise-fitting expression to a JSON file. The JSON record includes the target type, selected feature mode, feature names, number of fitted samples, expression string, pointwise loss, and symbolic complexity.

Solver-level selection. Pointwise fitting quality is not the final selection criterion. For each task, we reinsert candidate expressions into the corresponding optimizer and evaluate recovery performance on validation tasks. In graph learning, we form candidate objectives by combining edge and degree expressions,

$$\widehat{E}_{a,b}(w;z) = \sum_{e\in\mathcal{E}} \widehat{\rho}_a(w_e, z_e) + \sum_i \widehat{\eta}_b(d_i),$$

run projected gradient descent, and select the edge–degree pair with the best validation F1 among the evaluated combinations. The selected symbolic objective is then evaluated on held-out test tasks using the same projected solver.

In the spring-system setting, candidate symbolic gradients are inserted into the ISTA-style solver as the marginal correction from the learned penalty. Candidate expressions are evaluated with the same nonnegative sparse recovery solver and selected by validation F1 over a fixed threshold grid. This solver-level selection is used because a low component-level regression loss does not necessarily imply useful gradients, calibrated thresholds, or correct active-set behavior after optimization.

Protected symbolic operators. When symbolic expressions are reinserted into the solver, we evaluate logarithms and square roots using protected operators

$$\log_\varepsilon(x) = \log(\max\{x,\varepsilon\}), \qquad \sqrt{x}_\varepsilon = \sqrt{\max\{x,\varepsilon\}},$$

and clip the argument of the exponential before evaluation. These definitions match the solver-level candidate evaluator and keep the symbolic expressions real-valued along arbitrary projected rollouts.

A.5 Experimental Hyperparameters

This appendix summarizes the main hyperparameters used in the sparse recovery, spring-system, and graph-learning experiments. For each spring-system and graph-learning setting, we generate 10,000 training, 2,000 validation, and 2,000 held-out test instances. Each LP-ALISTA setting is trained separately, using 4096 fresh training examples per epoch and batches of 128. A fixed validation set of 1024 examples (signal/noise RNG seeds 1000/1001) is evaluated every epoch; checkpoints are selected by validation aggregate-SSE NMSE. The independent held-out test set has 1024 examples (signal/noise RNG seeds 2000/2001) and is not used for checkpoint selection or tuning.

Table 3: Hyperparameters for finite-budget sparse recovery experiments.

Hyperparameter	Value
Measurement dimension M	250
Expected sparsity S	50
Training epochs	$100 + 20K$
Initial learning rate	2×10^{-4}
Support-selection schedule	$(p_k)_{k=0}^{K-1} = \mathrm{int}(\mathrm{linspace}(\lfloor 1.2S/K \rfloor, 1.2S, K))$
Random seed	0

Table 4: Hyperparameters for spring-system learned penalty training and evaluation.

Hyperparameter	Value
Training epochs	500
Batch size	128
Training inner steps	30/100
Evaluation inner steps	30/100
ℓ_1 coefficient	5×10^{-3}
Step scale	2.0
Hidden dimension	64
Local MLP layers	2
Support positive weight	1.0

Table 5: Hyperparameters for spring-system symbolic distillation and solver-level selection.

Hyperparameter	Value
Symbolic target	Penalty gradient, `y_grad`
Grid size over s	17
PySR iterations	500
Population size	120
Maximum training samples	10,200
Maximum validation samples	5,000
Batching	Enabled
Pareto candidates evaluated	Up to 200
Top validation candidates	50
Selection metric	F1
Threshold grid	$0.01, 0.02, 0.03, 0.05, 0.07, 0.10, 0.15, 0.20$

Table 6: Hyperparameters for learned graph objective training.

Hyperparameter	$N_g = 20$	$N_g = 100$
Inner solver mode	Projected gradient	Projected gradient
Burn-in steps	50	50
Differentiable inner steps	150	150
Evaluation inner steps	200	200
Training epochs	500	500
Batch size	128	16
Outer learning rate	10^{-3}	10^{-3}
Inner step size η_{step}	10^{-3}	10^{-3}
Hidden dimension	64	64
MLP layers	1	1
Edge feature mode	Minimal	Minimal
Degree feature mode	Minimal	Minimal
Outer normalization	Enabled	Enabled

Table 7: Hyperparameters for graph objective symbolic regression and solver-level selection.

Hyperparameter	$N_g = 20$	$N_g = 100$
Maximum edge samples	1,000,000	10,000,000
Maximum degree samples	300,000	3,000,000
SR feature mode	Minimal	Minimal
Edge unary operators	$\sqrt{\cdot}, \log$	$\sqrt{\cdot}, \log$
Degree unary operators	$\log$	$\log$
Binary operators	$+, -, \times, /$	$+, -, \times, /$
PySR iterations	400	400
Population size	100	100
Maximum expression size	30	30
Batching	Enabled	Enabled
Edge candidates for solver selection	Top 10	Top 10
Degree candidates for solver selection	Top 10	Top 10
Maximum edge-degree combinations	100	100
Validation rollout steps	200	200
Test rollout steps	200	200
Selection metric	F1	F1

A.6 Compute Resources

Experiments were run on a local workstation with a single NVIDIA GeForce RTX 4090 GPU and on a remote server with four NVIDIA GeForce RTX 4090 GPUs. For one experimental setting, training the learned objective or penalty module takes approximately 90 minutes for finite-budget sparse recovery, 30 minutes for spring-system structure recovery, and 30 minutes for graph objective

discovery. A single symbolic regression run takes approximately 15 minutes. Runtime varies with the number of inner-solver steps, graph size, symbolic-regression search budget, and candidate-selection budget.

B Additional Results

B.1 Solver-Level Symbolic Selection

We analyze whether symbolic expressions should be selected by pointwise component fitting loss or by downstream solver behavior. This distinction is important because a symbolic expression with low component error can still induce poor gradients, incorrect active sets, or poorly calibrated edge weights when inserted back into the optimizer. We therefore compare three models: the neural teacher BOLD, the pointwise-best SR candidate selected by component fitting loss, and SR-BOLD selected by validation rollout performance.

Table 8: Effect of solver-level symbolic selection on graph learning from smooth signals, $N_g = 20$, RGG+Tikhonov setting. Pointwise-best SR denotes the symbolic candidate selected by component fitting loss, while solver-selected SR-BOLD is selected by validation rollout performance. Higher precision, recall, and F1 are better; lower error metrics are better.

Method	Precision	Recall	F1	Avg. Pred. Edges	Edge ℓ_1	Degree ℓ_1
BOLD teacher	**0.928**	0.951	0.935	20.49	**0.256**	**0.141**
Pointwise-best SR	0.496	**0.984**	0.617	58.05	1.042	0.343
Solver-selected SR-BOLD	0.925	0.962	**0.940**	20.89	0.267	0.176

Table 8 shows that pointwise symbolic fitting is not sufficient for recovering a useful optimization objective. The pointwise-best SR candidate obtains high recall but predicts almost three times as many edges as the ground-truth graph, resulting in low precision and poor F1. In contrast, solver-selected SR-BOLD closely matches the teacher's edge density and achieves a test F1 comparable to the neural teacher. This confirms that symbolic objectives must be evaluated in the solver loop, rather than selected only by component-level regression loss.

B.2 Feature-Prior Ablation

We study how the amount of analytic prior in the learned graph objective affects neural teacher performance and symbolic distillation. The experiment is conducted on the $N_g = 20$, RGG+Tikhonov setting. We compare three parameterizations of the factorized graph objective

$$E_\theta(w; z) = \sum_{(i,j)\in\mathcal{E}} \rho_\theta(f_{ij}) + \sum_i \eta_\theta(g_i),$$

where f_{ij} denotes edge-level features and g_i denotes node-level features.

The **Basic** parameterization provides strong analytic features related to the Kalofolias form:

$$f_{ij}^{\text{basic}} = [w_{ij},\, z_{ij},\, w_{ij}z_{ij},\, w_{ij}^2], \qquad g_i^{\text{basic}} = [d_i,\, \log(d_i + \varepsilon_{\text{deg}}),\, d_i^2].$$

The **Minimal** parameterization, used in the main experiments, removes these direct analytic-form features and only provides primitive optimization variables:

$$f_{ij}^{\min} = [w_{ij},\, z_{ij}], \qquad g_i^{\min} = [d_i].$$

The **Contextual** parameterization augments primitive variables with local graph context:

$$f_{ij}^{\text{ctx}} = [w_{ij},\, z_{ij},\, d_i + d_j,\, |d_i - d_j|], \qquad g_i^{\text{ctx}} = [d_i,\, c_i],$$

where

$$d_i = \sum_{e\in\mathcal{E}:\, i\in e} w_e, \qquad c_i = \frac{1}{N_g - 1} \sum_{j\neq i} z_{ij}.$$

All models are trained as neural teachers and then distilled into symbolic objectives using the same solver-level selection protocol.

Table 9: Feature-prior ablation on the $N_g = 20$, RGG+Tikhonov setting. The teacher columns report the neural learned objective, while the SR columns report the solver-selected symbolic objective. Higher F1 is better; lower edge and degree errors are better.

Parameterization	Teacher			SR-BOLD		
	F1 ↑	Edge ℓ_1 ↓	Degree ℓ_1 ↓	F1 ↑	Edge ℓ_1 ↓	Degree ℓ_1 ↓
Basic	0.956	0.217	0.136	**0.949**	0.542	0.288
Minimal	0.935	0.256	0.141	0.940	**0.267**	**0.176**
Contextual	**0.976**	**0.206**	**0.125**	0.847	0.880	0.485

Table 9 shows a trade-off between teacher expressivity and symbolic distillability. The Basic parameterization gives the teacher direct access to analytic ingredients of classical graph-learning objectives, such as wz, w^2, and $\log d$, and achieves strong neural performance. Its symbolic distillation also preserves support recovery well, but because the teacher already receives high-prior analytic features, this setting is less informative as evidence for objective discovery.

The Minimal parameterization removes these analytic-form features. Although its neural teacher is slightly weaker than Basic and Contextual, its symbolic objective remains close to the teacher and gives the best teacher-to-SR transfer among the three settings. This makes Minimal the most appropriate main setting: it uses substantially weaker analytic prior while remaining structured enough for reliable symbolic distillation.

The Contextual parameterization gives the strongest teacher, indicating that endpoint degree context and node signal context are useful for supervised objective learning. However, it is much harder to distill symbolically: SR-BOLD over-predicts edges and loses a large fraction of the teacher's support-recovery performance. This suggests that contextual teachers can learn powerful but more entangled objective components, making post-hoc symbolic regression substantially more difficult.

B.3 Burn-in and Differentiable-Tail Ablation

We ablate the burn-in strategy used for bilevel training of the graph-learning objective on the $N_g = 20$, RGG+Tikhonov setting. The total inner horizon is fixed to 200 steps. We vary how many initial steps are used as non-differentiated burn-in and how many final steps are kept as the differentiable tail. The burn-in steps reduce memory usage and avoid backpropagating through the full inner trajectory, while the differentiable tail still provides gradient information for updating the learned objective.

Table 10: Ablation of burn-in and differentiable-tail length for graph objective learning on $N_g = 20$, RGG+Tikhonov. The setting without burn-in becomes numerically unstable during training; we report the best checkpoint before divergence. Higher precision, recall, and F1 are better; lower error metrics are better.

Burn-in	Tail	Status	F1 ↑	Precision ↑	Recall ↑	Edge ℓ_1 ↓	Degree ℓ_1 ↓
0	200	NaN at epoch 6	0.917	0.891	0.947	1.032	0.323
50	150	Stable	**0.935**	**0.928**	**0.951**	**0.256**	**0.141**
100	100	Stable	0.933	0.923	0.937	0.282	0.171
150	50	Stable	0.930	0.919	0.935	0.306	0.198

Table 10 shows that differentiating through the full 200-step trajectory is unstable: training without burn-in produces NaNs after a few epochs, even though an early checkpoint can still recover a reasonable graph. Introducing burn-in stabilizes training and substantially improves edge and degree errors. Among the stable configurations, a moderate burn-in with a sufficiently long differentiable tail performs best. Increasing the burn-in further remains stable but slightly reduces recovery quality, suggesting that the differentiable tail still needs to be long enough to provide informative outer gradients. We therefore use burn-in plus differentiable tail as a practical compromise between memory cost, training stability, and bilevel gradient quality.

B.4 Long-Horizon Graph Objective Evaluation

We evaluate whether the learned graph objectives remain effective when optimized for more projected-gradient steps than the training/evaluation budget used in the main experiments. We consider two representative settings: $N_g = 20$, RGG+Tikhonov as a canonical small-scale setting, and $N_g = 100$, BA+Heat as a larger and more challenging graph-learning setting. For each setting, we evaluate the neural teacher BOLD and the solver-selected symbolic objective SR-BOLD at $K \in \{200, 500, 1000\}$ projected-gradient steps.

Table 11: Long-horizon evaluation of learned graph objectives. BOLD denotes the neural learned objective, and SR-BOLD denotes the symbolic objective selected by solver-level validation. Higher F1 is better; lower edge and degree errors are better.

Setting	K	Method	F1 ↑	Edge ℓ_1 ↓	Degree ℓ_1 ↓
$N_g = 20$, RGG+Tikhonov	200	BOLD	0.935	0.256	**0.141**
		SR-BOLD	**0.940**	0.267	0.176
	500	BOLD	0.932	**0.251**	0.142
		SR-BOLD	0.933	0.262	0.174
	1000	BOLD	0.935	0.253	0.145
		SR-BOLD	0.940	0.262	0.178
$N_g = 100$, BA+Heat	200	BOLD	0.904	0.411	0.107
		SR-BOLD	**0.916**	0.512	0.127
	500	BOLD	0.903	**0.279**	**0.103**
		SR-BOLD	0.911	0.304	0.128
	1000	BOLD	0.903	0.282	0.113
		SR-BOLD	0.911	0.312	0.135

Table 11 shows that both the neural and symbolic objectives remain stable under longer projected optimization. On $N_g = 20$, RGG+Tikhonov, the performance of BOLD and SR-BOLD is nearly unchanged from $K = 200$ to $K = 1000$, indicating that the learned objective is not merely tuned to a short rollout horizon. On $N_g = 100$, BA+Heat, increasing the number of steps improves edge error for both BOLD and SR-BOLD, while F1 remains stable. SR-BOLD consistently preserves the support-recovery behavior of the neural teacher and even slightly improves F1, although its edge and degree errors remain higher. Overall, these results support the interpretation that SR-BOLD is an executable long-horizon symbolic objective rather than a short-horizon surrogate.

B.5 All-Setting Generalization Stress Test

We further evaluate whether a single objective can be shared across all graph and signal families. This setting is substantially more heterogeneous than the per-setting experiments in the main text: graph densities, degree distributions, signal-distance scales, and optimal regularization balances vary across data-generating processes. We therefore view this experiment as a stress test for universal objective learning rather than as the primary benchmark.

For a fair comparison, the Kalofolias baseline is also tuned with a single global grid search over all settings, rather than separately for each graph/signal pair. We compare this globally tuned Kalofolias baseline with shared BOLD teachers and their solver-selected symbolic distillations. We consider two BOLD parameterizations: a *Minimal* teacher with low-prior inputs, and a *Contextual* teacher that additionally uses local graph and signal-context statistics.

Table 12: All-setting generalization stress test. Kalofolias is tuned once over all settings. BOLD uses a single shared learned objective across all graph and signal families. Higher F1 is better; lower edge and degree errors are better.

Method	Precision ↑	Recall ↑	F1 ↑	Edge ℓ_1 ↓	Degree ℓ_1 ↓
Kalofolias, global grid	0.695	0.818	0.719	0.741	0.272
BOLD, contextual	0.756	0.893	0.764	0.567	0.214
SR-BOLD, contextual	0.586	0.821	0.656	0.910	0.395
BOLD, minimal	0.713	0.774	0.707	0.691	0.275
SR-BOLD, minimal	0.678	0.793	0.699	0.799	0.349

Table 12 shows that a shared objective can still recover useful graph structure across heterogeneous settings, but the problem is considerably harder than per-setting training. The contextual teacher achieves the strongest validation performance and improves over the globally tuned Kalofolias baseline, suggesting that context statistics can help the learned objective adapt across data distributions. However, its symbolic distillation suffers a large performance drop, indicating that the contextual teacher learns a more entangled objective that is harder to represent with compact symbolic expressions. In contrast, the minimal teacher is weaker but much easier to distill: SR-BOLD remains close to the teacher in F1.

Selected symbolic objectives. The solver-selected symbolic objectives for the all-setting experiment are shown below. For the contextual objective, $d_{\mathrm{sum}} = d_i + d_j$ is the endpoint degree sum and c_i is the node signal context. For the minimal objective, the symbolic expression uses only the current edge weight w, signal distance z, and degree d.

All-setting SR-BOLD, contextual teacher

```
edge:
(z * -0.064828575) + ((sqrt(dsum) + (z / exp(z * 0.0640532))) * (w - (0.085140936 *
     (3.1458144 - dsum))))

degree:
(d / sqrt(c + 1.0784231)) - (c * 0.34432206)
```

All-setting SR-BOLD, minimal teacher

```
edge:
(z * 0.040983517) + (((log(z + exp(-0.202953 - exp(w))) * ((w - 0.2026189) * 3.4365442)
     ) + 0.23688596) / sqrt(sqrt(z + (log(z) * -2.3388636))))

degree:
(d - sqrt(d)) * 2.4859624
```

Discussion. The contextual symbolic objective contains explicit dependence on degree and signal-context statistics, which is consistent with the goal of adapting a shared objective to heterogeneous graph and signal distributions. However, the resulting SR-BOLD objective over-predicts edges and loses a substantial fraction of the teacher performance. This suggests that while contextual information is useful for the neural teacher, making such context-dependent objectives symbolically simple and solver-stable remains difficult.

The minimal symbolic objective is less expressive but transfers more reliably from teacher to symbolic form. Its degree component has a simple density-shaping form, while the edge component modulates edge costs by the signal distance and current edge estimate. Overall, the all-setting results support our limitation statement: learning one universal objective across heterogeneous data distributions remains challenging. Developing contextual teachers that are both adaptive and amenable to symbolic distillation is an important direction for future work.

B.6 Cross-Setting Objective Transfer

We evaluate direct transfer from the $N_g = 20$ RGG+Tikhonov source setting. The BOLD objective and the Kalofolias coefficients use the models and hyperparameters learned or tuned for that source setting and are applied to each target without retraining or retuning. For comparison, target-specific F1 uses the corresponding separately trained or tuned target-setting result.

Table 13: Direct transfer from the $N_g = 20$ RGG+Tikhonov source setting. Higher F1 is better; Δ F1 is transfer minus target-specific F1.

Target	Method	Transfer F1	Target-specific F1	Δ F1
RGG+Heat	Kalofolias	0.875	0.874	$+0.001$
RGG+Heat	BOLD	0.851	0.889	-0.038
BA+Tikhonov	Kalofolias	0.611	0.953	-0.342
BA+Tikhonov	BOLD	0.334	0.948	-0.614
BA+Heat	Kalofolias	0.615	0.844	-0.229
BA+Heat	BOLD	0.613	0.867	-0.254

Transfer is relatively mild from RGG+Tikhonov to RGG+Heat, where the graph family is unchanged and only the signal-generation mechanism changes. In contrast, transferring to BA graphs causes substantially larger degradation for both methods, with a particularly large drop for BOLD on BA+Tikhonov. These results indicate that the learned objectives are distribution-adapted rather than universally transferable, complementing the jointly trained all-setting stress test in Appendix B.5.

C Results of Symbolic Distillation

C.1 Symbolic Penalty Distillation on Spring Systems

This appendix reports the symbolic expressions distilled from the learned spring-system penalties. For each edge e, the learned penalty is decomposed as

$$P_\theta(s; q, v) = \sum_{e \in \mathcal{E}} \rho_\theta(s_e, \tilde{\phi}_e(q, v)),$$

where s_e is the current edge-weight estimate and

$$\phi_e(q, v) = [a_e,\, b_e,\, c_e,\, r_e] = [\mathrm{mean}_t\, |\Delta q_e(t)|,\, \mathrm{mean}_t\, \Delta q_e(t)^2,\, \mathrm{mean}_t\, |\Delta v_e(t)|,\, \mathrm{mean}_t\, \Delta v_e(t)^2].$$

The training-set feature normalizer maps these raw statistics to

$$\tilde{\phi}_e = [\tilde{a}_e, \tilde{b}_e, \tilde{c}_e, \tilde{r}_e] = \mathrm{Norm}(\phi_e).$$

Since the ISTA-style solver uses the derivative of the local penalty with respect to the current edge estimate, we distill symbolic expressions for the local marginal correction

$$\widehat{g}_e(s_e) \approx \frac{\partial \rho_\theta(s_e, \tilde{\phi}_e)}{\partial s_e}.$$

The corresponding energy can be obtained by integrating $\widehat{g}_e$ with respect to s_e, up to an additive constant independent of s_e. For consistency with the symbolic-regression implementation, the expressions below retain the variable names `dq_abs_mean`, `dq_sq_mean`, `dv_abs_mean`, and `dv_sq_mean`; in the exported dataset these names denote the corresponding normalized feature coordinates, not the raw trajectory statistics:

$$\texttt{s} = s_e, \quad \texttt{dq_abs_mean} = \tilde{a}_e, \quad \texttt{dq_sq_mean} = \tilde{b}_e, \quad \texttt{dv_abs_mean} = \tilde{c}_e, \quad \texttt{dv_sq_mean} = \tilde{r}_e.$$

Selected symbolic gradients. The following boxes list the selected symbolic gradients for each (N_p, K) setting. We display them in code-style boxes instead of standard display equations because the raw expressions are dense and easier to inspect with line wrapping.

Selected SR gradient, $N_p = 6, K = 30$

```
(((dq_abs_mean - (((dq_sq_mean + dv_abs_mean) + ((s - dq_abs_mean) * (dq_sq_mean -
    dq_abs_mean))) * -0.25109208)) - (s * 0.44216257)) - -1.6453906) * (((dq_abs_mean
    - 4.2924833) * ((s - dq_abs_mean) * 0.020622402)) + 0.3013461)
```

Selected SR gradient, $N_p = 6, K = 100$

```
(dq_sq_mean + ((s - 2.9219499) * (0.31393918 * (s - ((dq_sq_mean + dq_abs_mean) - (((((
    dq_abs_mean - -0.1971749) - (s * s)) * (dq_sq_mean - dq_abs_mean)) - dv_sq_mean) +
     -3.208588)))))) * 0.05575755
```

Selected SR gradient, $N_p = 20, K = 30$

```
(dq_sq_mean + ((((((s + dq_sq_mean) + (dq_abs_mean * ((dv_abs_mean * 0.36564818) -
    -0.77701294))) * 0.16328944) * ((dq_sq_mean * -0.43948478) - (((dv_sq_mean - (s *
    -4.278755)) * 0.16328944) + -4.278755))) + 3.0476546) + dv_abs_mean)) * 0.07960697
```

Selected SR gradient, $N_p = 20, K = 100$

```
((dq_abs_mean - (s - 2.768495)) * ((dq_sq_mean + (((s - dq_abs_mean) - dv_abs_mean) -
    3.4754918)) * -0.0072354437)) + (0.030808548 - (dq_sq_mean * -0.046767153))
```

C.2 Symbolic Objectives for Graph Learning from Smooth Signals

This appendix reports the symbolic objectives distilled from BOLD on representative graph learning from smooth signals settings. Unlike the spring-system experiments, where we distill the learned marginal correction, here we distill the decomposable *objective components* directly. This makes the recovered expressions comparable to the standard Kalofolias-style objective, which also consists of edge-level and degree-level terms. The expressions below are the raw PySR strings selected by solver-level validation. During solver execution, `log`, `sqrt`, and `exp` are evaluated using the protected definitions in Appendix A.4; the conventional notation below should therefore be read with those execution semantics.

The symbolic objective has the form

$$\widehat{E}(w; z) = \sum_{e=(i,j)\in\mathcal{E}} \widehat{\rho}(w_e, z_e) + \sum_i \widehat{\eta}(d_i), \qquad d_i = \sum_{e\in\mathcal{E}:\, i\in e} w_e.$$

Here w denotes the current edge weight, z denotes the signal distance, and d denotes the node degree. The expressions below are the raw symbolic-regression outputs selected by solver-level validation. We present them in code-style boxes because the formulas are dense and easier to inspect with line wrapping.

Selected Expressions

$N_g = 20$**, RGG + Tikhonov**

```
edge:
((sqrt(z) * ((z * -0.2065593) + w)) - (w * (w * (10.781507 - z)))) - -0.17112947

degree:
(d + -1.5957252) / ((2.1447394 / ((d * (((d * d) - d) - -1.5965252)) - -2.018577)) -
    -0.5801557)
```

$N_g = 20$, RGG + Heat

```
edge:
(((((z + -14.033947) * w) + 0.62100214) + sqrt(z)) * w) + (((sqrt(z) * -0.16890372) * z
    ) - -0.18299527)

degree:
(d + -1.7191416) / ((-1.5067106 / ((d * (((((d * d) + -0.9531056) / (-1.3373872 / log(d
    ))) + -1.2071054) * d)) - 1.022277)) + 0.33457175)
```

$N_g = 20$, BA + Tikhonov

```
edge:
(((w * -9.044619) - -0.42098165) * (w + 0.606018)) + (((log(z) * -0.7162198) + (w *
    3.1310434)) * z)

degree:
((2.6353736 - (-31.179699 / (-10.811896 - ((d * (d * d)) * d)))) * (d + -1.4975227)) -
    0.45004183
```

$N_g = 20$, BA + Heat

```
edge:
((((z * 0.10098137) - w) * ((sqrt(w + z) + log(0.75460863 - w)) / -0.24523498)) - (w /
    0.17029725)) + 0.5808955

degree:
((d + -0.98543394) / ((1.6403452 / ((d * ((d + -0.6634944) * (d - 1.6354425))) * d)) +
    0.64571583)) - 0.34750983
```

$N_g = 100$, RGG + Tikhonov

```
edge:
sqrt(z) + ((0.36811587 - z) - (w * (w / (sqrt(sqrt(z)) / 12.749084))))

degree:
(((d * -0.8576137) + (log(d) * (log(d) + (d * 0.14373276)))) * d) * log(d)
```

$N_g = 100$, RGG + Heat

```
edge:
((w * (log(sqrt(z)) + (w * -10.766033))) - (z / 2.6917315)) + 0.52474695

degree:
(d - -2.141827) * ((-411.0971 / ((d * d) + 165.01471)) + 1.9086748)
```

$N_g = 100$, BA + Tikhonov

```
edge:
((w * w) * -37.944973) + ((z * 2.0395672) * ((w * z) - (z + -1.1181854)))

degree:
(((d * ((d * 8.068839e-5) + -0.008888539)) + 0.31817228) * (d + -7.0796375)) * (d -
    1.1607461)
```

$N_g = 100$, **BA + Heat**

```
edge:
((w * -34.937286) * w) - ((z * ((z - ((z - sqrt(w)) * 0.35448557)) - (w - (-0.71783626
    - w)))) * 2.212563)

degree:
((d + -5.27017) / (0.4374437 - (((-33.83603 / d) + 0.80715823) / d))) - 1.5279919
```