GENERATIVE POLICY IMPROVEMENT FROM OBSERVATIONAL DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

We study how to improve decision policies from observational data to *generate* better actions when actions are complex, structured objects, such as text messages or images. This task is subject to two major challenges: (i) action-level overlap is often too weak for reliable off-policy evaluation and improvement, and (ii) generating new structured actions often requires expensive training or fine-tuning (e.g., of an LLM) and can be ill-defined due to information loss in action representations. To address both, we propose POLICYLIFT, a three-stage framework for *generative policy improvement* with structured actions from observational data. In Stage 1, POLICYLIFT learns a small set of discrete action codes that capture high-level strategies in the observed action space and are tailored to support policy improvement under limited overlap. In Stage 2, it learns a context-dependent policy over these action codes. At deployment time, Stage 3 "lifts" the learned code-level policy back to the original action space by generating actions from the historical behavior system in a code-conditioned way. We show theoretically that improving the policy over action codes guarantees improvement of the induced policy in the original action space. We demonstrate POLICYLIFT across experiments in synthetic and real-world text generation tasks with consistent policy improvements over the behavior policy.

1 INTRODUCTION

Learning decision policies from observational data, i.e., off-policy learning (OPL), asks how to improve a behavior policy using logged interactions rather than new online experiments (Athey & Wager, 2021; Dudík et al., 2014; Kallus et al., 2022; Swaminathan & Joachims, 2015). Existing work studies off-policy evaluation (OPE) and OPL primarily under relatively simple action spaces, such as binary, categorical, or low-dimensional interventions, where actions correspond to choices like whether to treat a patient, which product to recommend, or which price to offer (Kitagawa & Tetenov, 2018; Athey & Wager, 2021; Kallus et al., 2022). In many modern systems, actions are *complex* and *high-dimensional*; for example, an online platform can generate a personalized headline or advertisement (Matias et al., 2021; Meguellati et al., 2024) or a mental health support system may suggest an empathic reply in an online peer-support conversation (Sharma et al., 2023).

We study OPL for such complex action spaces. Given logged triples (X, A, Y) from a behavior policy, where X is a context, A is an action, and Y is an observed outcome, the goal is to learn a target policy that can *generate* improved structured actions at deployment time. In online mental health support, X may encode the help-seeker's post and conversation history, A is a generated or suggested supportive response, and Y measures empathy, engagement, or conversation quality. In headline optimization, X may encode an article excerpt, A is a generated headline, and Y is reader engagement.

However, such complex, high-dimensional actions are often implicitly generated by a latent lower-dimensional structure, such as how style and content might be used to generate a specific supportive exercise suggestion. In these settings, policy improvement requires not selecting over a small discrete set of actions, but *generating* high-dimensional, complex actions like such messages. Two **challenges** make this substantially harder than standard off-policy learning:

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.

① ***Limited overlap.*** OPE and OPL require overlap (Dudík et al., 2014; Kallus et al., 2022), which means that the behavior policy covers the actions a new target policy would select. For complex actions, this is often implausible: the behavior system concentrates on a vanishingly small subset of the action space for each context. Even when the behavior system is stochastic, the randomness is often local and thus does not induce meaningful coverage across qualitatively different actions. Existing approaches to limited overlap either constrain the policy class (Kallus, 2021; Schweisthal et al., 2023; Zhao et al., 2024) or optimize conservative objectives (Jin et al., 2025), but these operate in the original action space and cannot resolve the fundamental coverage gap in high-dimensional structured spaces.

② ***Information loss.*** It is unclear a priori how to learn action representations that both make overlap plausible and still allow mapping coarse latent actions back to raw actions. A representation-level decision (e.g., a text style) is not yet a deployable action: deployment still requires a specific response in the original action space (e.g., natural language). Prior work learns improved policies in a subspace of the actions (Saito & Joachims, 2022; Peng et al., 2023), but if the representation does not fully mediate the effect of the action on the outcome (Saito et al., 2023), instantiating raw actions (e.g., via a decoder) can induce a systematic mismatch between the value optimized in representation space and the value delivered in the original action space. We formalize this later as a *deployment gap*.

To address these challenges, we propose POLICYLIFT, a framework for *generative* policy improvement from observational data. POLICYLIFT first maps each observed action to a discrete *action code*, where *codes* represent high-level action strategies that are easier to compare, estimate, and optimize under limited overlap. Next, it learns a context-dependent policy over these codes using standard off-policy learning tools. Finally, it lifts the learned code-level policy back to the original action space by using the behavior system itself as a generator. We call this *behavior-consistent lifting* (BCL) and show that, when the behavior system is sampled until an action with the target code is found, BCL eliminates the deployment gap, so improvements in true code-level value transfer exactly to the deployed policy over structured actions.

Our contributions are the following. **(i)** We formulate OPL with *generative* structured actions, where the target policy must generate new actions rather than select from a fixed action set, and propose POLICYLIFT to address this setting. **(ii)** We derive various theoretical guarantees for our framework. In particular, we prove that improving the policy over action codes guarantees improvement of the induced policy in the original action space. **(iii)** We empirically validate our theory and show that POLICYLIFT improves over baselines in LLM-based text generation tasks.

2 RELATED WORK

Table 1: *Legend.* ✓ = addressed; ✗ = not addressed; (✓) = partially addressed; (✗) = assumed or not enforced; – = not applicable.

	Unseen structured actions	Learned repr.	Overlap controlled	Learns policy	Generative deployment
Overlap-constrained OPL (Kallus, 2021; Schweisthal et al., 2023; Zhao et al., 2024; Jin et al., 2025)	✗	–	✓	✓	–
Action embeddings / MIPS (Saito & Joachims, 2022; Saito et al., 2023)	✗	(✓)	(✗)	✗	✗
PoTEC (Saito et al., 2025)	✗	(✓)	(✗)	✓	✗
Learned action embeddings (Peng et al., 2023; Cief et al., 2024)	✗	✓	(✗)	✗	✗
Prompt-guided generation (Kiyohara et al., 2025)	(✓)	✗	(✗)	✓	✓
POLICYLIFT (ours)	✓	✓	✓	✓	✓

Policy learning from observational data. OPL studies how to improve a behavior policy from logged interactions without collecting new online data. Standard approaches estimate or optimize policy value using inverse propensity weighting, doubly robust estimation, or regression adjustment (Swaminathan & Joachims, 2015; Dudík et al., 2014; Kallus et al., 2022; Athey & Wager, 2021). A separate stream addresses limited overlap by modifying the policy-learning problem itself; for example, by restricting, retargeting, or otherwise regularizing the learned policy toward better-supported interventions (Kallus, 2021; Schweisthal et al., 2023; Zhao et al., 2024), or using pessimistic value criteria that penalize policies with high uncertainty under the logged data (Jin et al., 2025; Swaminathan & Joachims, 2015). *However, these approaches operate in simple or low-dimensional action spaces and do not address policy learning with structured actions.*

OPE/OPL with complex action spaces (an extended related work is in Appendix B). Several works extend OPE and OPL beyond binary or categorical actions. For continuous actions, semiparametric and doubly robust methods learn policies without discretizing the action space (Kallus & Zhou, 2018; Demirer et al., 2019). For OPE in large action spaces, action-embedding methods replace action-level propensities with marginalized propensities (MIPS) over embeddings to reduce variance (Saito & Joachims, 2022), potentially also accounting for residual variance within clusters (OffCEM, (Saito

et al., 2023)). PoTEC extends this idea to OPL by decomposing the policy into cluster selection and within-cluster action selection (Saito et al., 2025), using pre-defined clusters or a separate method to a priori perform action clustering (e.g. Peng et al., 2023). *These methods operate on a fixed, enumerable action set, and do not extend to the generation of new structured actions.* Other approaches learn action groups or action embeddings from logged data (Peng et al., 2023; Cief et al., 2024). *However, these methods target OPE, not OPL, and assume sufficient overlap in the learned representation. In contrast,* POLICYLIFT *learns action codes that encourage overlap, optimizes a policy over those codes, and generates code-conditional structured actions by sampling from the behavior system.*

Closest to our setting is Kiyohara et al. (2025), which studies OPL for sentence generation, but where the logged action is an observed discrete prompt and deployment uses a frozen generator (behavior system) to map prompts to text. *In contrast,* POLICYLIFT *learns a code policy from logged structured actions* ***without*** *requiring logged prompts, and deploys it using only sampling access to the behavior system.*

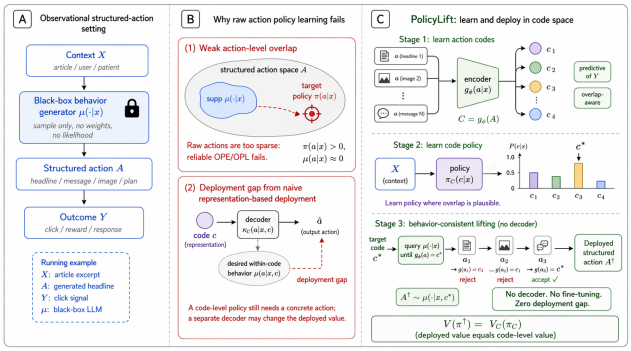


Figure 1: **Overview.** Generative policy improvement from observational data with complex, structured actions.

3 PROBLEM SETUP

Off-policy learning. We consider the standard setup for policy learning from observational data (Dudík et al., 2014; Athey & Wager, 2021; Kallus et al., 2022). We observe i.i.d. data $\mathcal{D} = \{(X_i, A_i, Y_i)\}_{i=1}^n$ generated by a historical *behavior policy* $\mu(\cdot \mid x)$, where $X \in \mathcal{X}$ is a context, $A \in \mathcal{A}$ is a structured action such as an LLM-generated text, and $Y \in \mathbb{R}$ is an outcome. Importantly, we do *not* assume a fixed pre-defined set of finite actions. We let $Y(a)$ denote the potential outcome under action a (Rubin, 1974). For any policy $\pi(\cdot \mid x)$ over $\mathcal{A}$, the *policy value* is $V(\pi) := \mathbb{E}[Y(\tilde{A})],\ \tilde{A} \sim \pi(\cdot \mid X)$. In addition to logged data, we assume sampling access to the behavior policy at deployment time: given a context x, we can draw candidate structured actions from $\mu(\cdot \mid x)$, but we do not require access to its weights, likelihoods, or training data. This captures settings where the historical behavior system is available as a black-box generator. Our goal is to learn a target policy that can *generate complex raw actions* at deployment.

Identification. We adopt the standard identification assumptions for OPL (Rubin, 1974; Athey & Wager, 2021), namely, *consistency* (i.e., $Y = Y(A)$) and *ignorability* (i.e., $Y(a) \perp A \mid X$ for all $a \in \mathcal{A}$). Under these assumptions, the conditional expected potential outcome is identified by the regression function $m(x, a) := \mathbb{E}[Y(a) \mid X = x] = \mathbb{E}[Y \mid X = x, A = a]$, and the value of any target policy satisfies $V(\pi) = \mathbb{E}[m(X, \tilde{A})]$ with $\tilde{A} \sim \pi(\cdot \mid X)$. In addition, identification of $V(\pi)$ for a target policy π requires *overlap* (also called positivity or common support) (Rosenbaum & Rubin, 1983; Imbens & Rubin, 2015; D'Amour et al., 2021), namely, the behavior policy covers the support of π; i.e., for P_X-almost all $x \in \mathcal{X}$, $\mu(a \mid x) > 0$ whenever $\pi(a \mid x) > 0$. Overlap violations bias value estimates (Felicioni et al., 2022; Sachdeva et al., 2020), and weak overlap inflates the variance of importance-weighted estimators (Saito & Joachims, 2022; Sachdeva et al., 2024; Crump et al., 2009) and the extrapolation error of regression-based ones (Farajtabar et al., 2018; Voloshin et al., 2021).

4 POLICYLIFT FRAMEWORK

Overview: We propose POLICYLIFT, a three-stage *generative* framework for *policy improvement* from observational data over structured action spaces. Figure 1 shows the framework, which separates policy learning from action generation. Briefly, in **Stage** (1), we encode each observed action into a discrete action code $C = g_\phi(A) \in \{1, \ldots, K\}$ via a tailored learned encoder. This maps the high-dimensional structured action space $\mathcal{A}$ to a low-dimensional code space where overlap is more plausible. In **Stage** (2), we learn a context-dependent code-level policy $\pi_C(c \mid x)$ from the induced data (X_i, C_i, Y_i). In **Stage** (3), we deploy π_C through our so-called *behavior-consistent*

lifting: for a target code sampled from $\pi_C(\cdot \mid x)$, we draw candidate actions from the behavior system $\mu(\cdot \mid x)$ and return an action whose code matches the target. This makes the learned policy *generative*: POLICYLIFT outputs a deployed policy sampler $\pi^{\uparrow}(\cdot \mid x)$ over original structured actions. Algorithm 1 shows the pseudocode.

Stage ①: Learning action codes. Stage 1 learns an encoder $g_\phi : \mathcal{A} \to \{1, \dots, K\}$ that maps each action to a discrete *action code* $C = g_\phi(A)$, with induced code-level behavior policy $\mu_C(c \mid x) := \Pr(C = c \mid X = x)$. Encoder learning targets two properties that determine the value of the code-level policy learning in Stage 2, namely the code should be (i) *predictive* of the outcome, and (ii) the code-conditional context distribution should *overlap* with the marginal. We describe both below:

• **(i) *Predictiveness.*** The code C should be predictive of Y given X, i.e., the code regression function $m_C(x, c) := \mathbb{E}[Y \mid X = x, C = c]$ should vary meaningfully across c. If C carries no information about Y beyond X, no code-level policy can improve over μ. Thus, the underlying variation determines how much a code-level policy can improve over the behavior policy.

• **(ii) *Overlap.*** The code-level overlap assumption needed for policy learning and deployment is formulated wrt. the behavior code propensity $\mu_C(c \mid x)$. Stage 1 uses a distributional surrogate for this assumption instead: the code-conditional context distribution $Q_c^\phi := P(X \mid C = c)$ should be close to the marginal P_X for each code c. These two notions are related by Bayes' rule. Let $p_C^\phi(c) := \Pr(C = c)$. Whenever $Q_c^\phi \ll P_X$ (i.e., Q_c^ϕ is absolutely continuous with respect to P_X, so it assigns no mass outside the support of P_X), we aim for $\mu_C(c \mid x) = p_C^\phi(c) \frac{\mathrm{d}Q_c^\phi}{\mathrm{d}P_X}(x)$ for P_X-almost all x. Thus, in the ideal balanced case $Q_c^\phi = P_X$, the code propensity is context-independent, $\mu_C(c \mid x) = p_C^\phi(c)$. More generally, distributional balance controls how much code propensities vary with x, while marginal code balance controls whether any code is globally rare. This balance reduces extrapolation when code-level outcome regressors are evaluated under the marginal context distribution.

Objective. We learn the action encoding g_ϕ by minimizing a trade-off between the two properties,

$$\min_{\phi,\theta} \quad \underbrace{\frac{1}{n}\sum_{i=1}^{n}\big(Y_i - f_\theta(X_i, g_\phi(A_i))\big)^2}_{\text{predictiveness}} + \beta \underbrace{\sum_{c=1}^{K} \widehat{D}\big(\widehat{P}_X, \widehat{Q}_c^\phi\big)}_{\text{distributional overlap}} + \gamma \underbrace{\mathrm{KL}\big(\widehat{p}_C^\phi \,\|\, \mathrm{Unif}_K\big)}_{\text{marginal code balance}}. \tag{1}$$

Here f_θ is an outcome predictor, $\widehat{D}$ is an empirical distributional distance such as sliced Wasserstein, and $\widehat{p}_C^\phi$ is the empirical marginal distribution of learned codes. The trade-off weight $\beta \geq 0$ balances attainable value against finite-sample reliability, while $\gamma \geq 0$ prevents globally rare or empty codes (and a trivial single-code minimzer of the overlap term).

Stage ②: Learning a code-level policy. Given the encoder from Stage 1, we re-encode the logged actions as $C_i = g_\phi(A_i)$ and learn a code-level policy $\pi_C(\cdot \mid x)$ on (X_i, C_i, Y_i).

• *Feasible policy class.* For a threshold $\tau > 0$, we restrict to

$$\Pi_{C,\tau} := \{\pi_C : \pi_C(c \mid x) > 0 \Rightarrow \mu_C(c \mid x) \geq \tau\}. \tag{2}$$

This constraint is the pointwise code-level overlap condition used for policy learning and deployment. The Stage 1 distributional and marginal-balance regularizers make the constraint easier to satisfy by encouraging code propensities to be less context-dependent and not globally rare. However, these regularizers are only surrogates: distributional balance alone does not imply a uniform lower bound on $\mu_C(c \mid x)$.

• *Objective.* We learn $\pi_C \in \Pi_{C,\tau}$ by maximizing a regression-adjusted (RA) value estimator

$$\widehat{V}_{\mathrm{RA}}(\pi_C) := \frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{K} \pi_C(c \mid X_i)\, \widehat{m}(X_i, c), \tag{3}$$

where $\widehat{m}(x, c)$ is a learned regressor for $m_C(x, c)$. We use RA rather than IPW or DR estimators because Stage 1 directly regularizes the context regions on which the code-level regressors are evaluated (Theorem (5.6)).

Stage (3): Behavior-consistent lifting. The learned code-level policy π_C is not itself a deployable policy yet: at deployment time, the system must output a concrete action $a \in \mathcal{A}$. We propose a simple but effective solution as follows. For a context x, Stage 3 first selects a target code given the learned code policy $c^\star \sim \pi_C(\cdot \mid x)$, then generates an action whose code is $c^\star$ by *behavior-consistent lifting*: we sample $a \sim \mu(\cdot \mid x)$ and accept the first draw with $g_\phi(a) = c^\star$. In practice, we cap the number of draws at $T_{\max}$ and fall back to $\mu(\cdot \mid x)$ if no match is found. The threshold τ also controls deployment cost: for supported target codes, the expected number of draws scales as $1/\mu_C(c^\star \mid x)$, and the feasible class keeps this quantity bounded.

Algorithm 1: POLICYLIFT

Input: logged data $\mathcal{D} = \{(X_i, A_i, Y_i)\}_{i=1}^n$; number of codes K; overlap weights $\beta \geq 0, \gamma \geq 0$; threshold $\tau > 0$; draw budget $T_{\max}$.
Output: deployed policy sampler $\widehat{\pi}^{\uparrow}(\cdot \mid x)$.

Stage (1): Learn action codes
Fit encoder $\widehat{g}_\phi$ and outcome predictor f_θ
Re-encode logged actions: $C_i \leftarrow \widehat{g}_\phi(A_i)$

Stage (2): Learn code-level policy
Fit code-level regression $\widehat{m}(x, c) \approx m_C(x, c)$ on $\{(X_i, C_i, Y_i)\}_{i=1}^n$
Solve $\widehat{\pi}_C \in \arg\max_{\pi_C \in \Pi_{C,\tau}} \widehat{V}_{\mathrm{RA}}(\pi_C)$

Stage (3): Deployment at context x
Sample target code $c^\star \sim \widehat{\pi}_C(\cdot \mid x)$
for $t = 1, \ldots, T_{\max}$ **do**
  Sample $a_t \sim \mu(\cdot \mid x)$
  if $\widehat{g}_\phi(a_t) = c^\star$ **then**
    return a_t
return an independent draw from $\mu(\cdot \mid x)$

5 THEORETICAL ANALYSIS

We now analyze three key properties of POLICYLIFT: (1) *behavior-consistent lifting* makes a code-level policy deployable in the original action space without changing its value; (2) *predictiveness of action codes* determines the attainable improvement of code-level policy learning; and (3) *regression-adjusted policy learning* with balanced action codes to reduce extrapolation error. Together, these results justify the three design choices of POLICYLIFT: behavior-consistent lifting, predictive code learning, and overlap regularization. All proofs are in Appendix C.

5.1 BEHAVIOR-CONSISTENT LIFTING

We first analyze **Stage (3)**. The key quantity is the potential value difference between the code-level policy and the deployed action-level policy. We define an *instantiation rule* (i.e., decoder or generator) as a conditional distribution $\kappa(a \mid x, c)$ over actions given a context-code pair. Combined with a code policy π_C, it induces a deployed policy on actions (Saito et al., 2025) $\pi^\kappa(a \mid x) := \sum_{c=1}^K \pi_C(c \mid x)\, \kappa(a \mid x, c)$.

Behavior-consistent lifting corresponds to $\kappa(\cdot \mid x, c) = \mu(\cdot \mid x, c)$, the within-code behavior distribution. We denote the resulting deployed policy by

$$\pi^{\uparrow}(a \mid x) := \sum_{c=1}^{K} \pi_C(c \mid x)\, \mu(a \mid x, c). \tag{4}$$

The acceptance loop used by POLICYLIFT implements this instantiation rule exactly in the ideal uncapped case: Proposition C.4 in Appendix C.4 shows that accepting the first draw from $\mu(\cdot \mid x)$ whose code equals c returns a sample from $\mu(\cdot \mid x, c)$. Thus, behavior-consistent lifting is *not* an additional learned decoder, but *a sampler of the within-code behavior distribution*.

Theorem 5.1 (Value under code-based deployment)**.** *Under consistency and ignorability, for any code policy π_C and instantiation rule κ, the policy value at deployment is given by*

$$V(\pi_\kappa) = \underbrace{V_C(\pi_C)}_{\textit{code-level value}} + \underbrace{\mathbb{E}\left[\sum_{c=1}^{K} \pi_C(c \mid X) \Delta_\kappa(X, c)\right]}_{\textit{deployment gap}}, \tag{5}$$

where $V_C(\pi_C) := \mathbb{E}[\sum_c \pi_C(c \mid X)\, m_C(X, c)]$, $\Delta_\kappa(x, c) := \mathbb{E}_{A \sim \kappa(\cdot \mid x, c)}[m(x, A)] - \mathbb{E}_{A \sim \mu(\cdot \mid x, c)}[m(x, A)]$. In particular, $\Delta_\kappa(x, c) = 0$ if $\kappa(\cdot \mid x, c) = \mu(\cdot \mid x, c)$, or if $m(x, a)$ is constant over actions receiving mass within code c.[1]

Intuitively, Theorem 5.1 decomposes deployed value into (i) the code-level policy value $V_C(\pi_C)$ and (ii) the *deployment gap*, which measures how the instantiation rule κ changes the within-code action

[1]This condition is analogous to Assumption 3.2 of Saito & Joachims (2022): the action embedding fully mediates the effect of the action on the outcome.

distribution relative to the behavior system. The deployment gap can be nonzero whenever κ reweights actions within a code in a way that changes their expected outcomes.[2] Our behavior-consistent lifting avoids this issue by setting $\kappa(\cdot \mid x, c) = \mu(\cdot \mid x, c)$ directly. Consequently, Theorem 5.1 gives $\Delta_\kappa \equiv 0$, so the deployment gap vanishes: the value optimized at the code level matches exactly the value achieved after deployment, as shown in the following corollary:

Corollary 5.2 (Value equivalence)**.** *Under consistency and ignorability, for any π_C satisfying code-level overlap, the behavior-consistent deployed policy from Eq.* (4) *satisfies $V(\pi^\uparrow) = V_C(\pi_C)$.*

Corollary 5.2 makes code-level policy learning a valid deployment target: a policy can be optimized and evaluated in the simpler code space, and *the same improvement value is realized when the policy is lifted back to the original structured action space*. In particular, any code-level improvement transfers directly to a deployed improvement over the behavior policy in the original action space.

Corollary 5.3 (Improvement transfer)**.** *Under the conditions of Corollary 5.2: if $V_C(\pi_C) \geq V_C(\mu_C)$, then $V(\pi^\uparrow) \geq V(\mu)$.*

Behavior-consistent lifting also has bounded expected sampling costs. For a target code $c^\star$, the number of behavior-system samples required for acceptance is controlled by the code propensity $\mu_C(c^\star \mid x)$. Therefore, the overlap threshold τ ensures code-level support while also bounding the expected number of behavior-system samples required in expectation, with $T_{\max}$ providing an additional safeguard on worst-case cost.

Lemma 5.4 (Sampling cost)**.** *For a target code $c^\star$ at context x, the expected number of draws from $\mu(\cdot \mid x)$ until $g_\phi(a) = c^\star$ is $1/\mu_C(c^\star \mid x)$. For $\pi_C \in \Pi_{C,\tau}$, this is bounded by $1/\tau$.*

5.2 Predictiveness and the attainable code-level value

We next lower-bound the attainable code-level policy value in terms of the predictive power of C. Let $\bar{m}(x) := \mathbb{E}[Y \mid X = x]$ denote the context-only regression. The baseline value of the behavior policy is $V(\mu) = \mathbb{E}[\bar{m}(X)]$. The best attainable code-level value under encoder g_ϕ is the oracle value

$$V_C^\star(\phi) := \sup_{\pi_C} \mathbb{E}\left[\sum_{c=1}^{K} \pi_C(c \mid X)\, m_C(X, c)\right] = \mathbb{E}\left[\max_{c \in \{1,\dots,K\}} m_C(X, c)\right], \tag{6}$$

where the second equality holds because the inner objective is linear in $\pi_C(\cdot \mid x)$. We define the predictiveness signal of the encoder as the conditional variance of the code regression, i.e., $\mathrm{Sig}(\phi) := \mathbb{E}[\mathrm{Var}(m_C(X, C) \mid X)]$. If C is conditionally constant given X, then $m_C(X, C) = \bar{m}(X)$ almost surely and $\mathrm{Sig}(\phi) = 0$; in that case, we have $V_C^\star(\phi) = V(\mu)$, and no code-level policy can improve over the behavior policy.

Theorem 5.5 (Predictiveness implies value potential)**.** *Assume $|Y| \leq B$ a.s. for some $B > 0$. Then,* $\mathrm{Sig}(\phi)$ *admits the equivalent characterizations*

$$\mathrm{Sig}(\phi) = \mathbb{E}\big[(m_C(X, C) - \bar{m}(X))^2\big] = \mathbb{E}\big[(Y - \bar{m}(X))^2\big] - \mathbb{E}\big[(Y - m_C(X, C))^2\big], \tag{7}$$

and the oracle value in Eq. (6) *satisfies the lower bound*

$$V_C^\star(\phi) \;\geq\; V(\mu) \;+\; \frac{1}{2B}\,\mathrm{Sig}(\phi). \tag{8}$$

The second equality in Eq. (7) identifies $\mathrm{Sig}(\phi)$ as the reduction in optimal prediction error when predicting Y from (X, C) instead of from X alone. Learning g_ϕ to make the induced code $C = g_\phi(A)$ predictive of Y, i.e., to reduce the MSE of predicting Y from (X, C), therefore increases $\mathrm{Sig}(\phi)$. The inequality in Eq. (8) then translates this signal into attainable improvement over the behavior policy. $\Rightarrow$ *This directly relates to the predictiveness term in the* ***Stage*** (1) *loss.*

Theorem 5.5 has a different focus from existing OPE analyses for action embeddings or clusters (Saito & Joachims, 2022; Saito et al., 2023; Peng et al., 2023). Existing analyses characterize estimator bias, variance, or MSE under embedding-mediation, local-correctness, or within-group outcome-similarity conditions; here, we lower-bound the value attainable by code-level policy learning in terms of an MSE-reduction signal.

[2]The deployment gap can be positive if the instantiation rule improves over the code-level policy value; PoTEC (Saito et al., 2025) uses a two-stage OPL approach that exploits this for a finite, enumerable action set. However, in complex structured action spaces such as text, new actions must be generated for a new context x. The key challenge is that κ can then place mass outside the within-code support of μ, making the deployed value **not** identified from logged data when this occurs with positive probability under the learned policy (Proposition C.2). The deployed policy can then be even *worse* than the original behavior policy.

5.3 Overlap and RA estimation error

Recall that code policy learning proceeds by first estimating the outcome regressor $\hat{m}(\cdot, c)$ and then producing a code-level policy by maximizing (or soft-maximizing) this function for a given input context. The central challenge for regression is extrapolation. The code regressor $\widehat{m}(\cdot, c)$ is fit on contexts drawn from Q_c^ϕ, but the RA value estimator from Eq. (3) integrates $\widehat{m}$ against the full marginal P_X weighted by $\pi_C(\cdot \mid x)$. If Q_c^ϕ is far from P_X for some code, learning $\widehat{m}(\cdot, c)$ requires extrapolating to context regions with few or no logged observations under code c. Let $e_c(x) := \widehat{m}(x, c) - m_C(x, c)$ denote the prediction error. To quantify how this extrapolation inflates the estimation error of $\widehat{V}_{\mathrm{RA}}$, we posit the following shift bound, which relates prediction error under P_X to prediction error under Q_c^ϕ:

$$\mathbb{E}_{P_X}[e_c(X)^2] \;\le\; \mathbb{E}_{Q_c^\phi}[e_c(X)^2] \;+\; \lambda_{\mathrm{ov}}\, D(P_X, Q_c^\phi), \tag{9}$$

for some distance D on distributions over $\mathcal{X}$ and a constant $\lambda_{\mathrm{ov}} > 0$.[3] Recall $V_C(\pi_C)$ is the true value of π_C. We can upper bound the RA estimation error into the prediction error on the code-conditional distribution, by an overlap penalty measuring the total distance between P_X and the code-conditional distributions Q_c^ϕ, and a standard sample-complexity term:

Theorem 5.6 (RA estimation error under limited overlap)**.** *Assume $|Y| \le B$ a.s. and $|\widehat{m}(x, c)| \le B$ for all (x, c). Let $\widehat{m}$ be independent of the evaluation sample (e.g., via cross-fitting), let Π_C be a finite code-policy class, and assume Eq.* (9) *holds. Then, with probability at least $1 - \delta$, simultaneously for all $\pi_C \in \Pi_C$, it holds that*

$$\left|V_C(\pi_C) - \widehat{V}_{\mathrm{RA}}(\pi_C)\right| \;\le\; \sqrt{\sum_{c=1}^{K} \mathbb{E}_{Q_c^\phi}\!\left[e_c(X)^2\right] \;+\; \lambda_{\mathrm{ov}} \sum_{c=1}^{K} D(P_X, Q_c^\phi)} + \; B\sqrt{\frac{2\log(2|\Pi_C|/\delta)}{n}}. \tag{10}$$

Note the overlap penalty is propensity-free: $D(P_X, Q_c^\phi)$ depends only on the empirical distributions of X and $X \mid C = c$, so it can be minimized directly during representation learning. This justifies the second term of the **Stage** ① loss. Theorem 5.6 is unlike prior work on OPE under limited overlap, which focuses on variance inflation of importance-weighted estimators (Crump et al., 2009; Dudík et al., 2014; Farajtabar et al., 2018; Swaminathan & Joachims, 2015; Voloshin et al., 2021); in contrast, our theorem characterizes RA extrapolation error and adapts the balanced-representation bound (Johansson et al., 2016; Shalit et al., 2017) to learned discrete codes and code-level policy value.

6 Neural implementation

We describe a neural implementation of PolicyLift. We provide a full pseudocode with implementation choices in Appendix H in Alg. 2.

Stage ①: Learning action codes. For learning the encoder g_ϕ, we use a two-branch network. The *action branch* maps A to logits over $\{1, \dots, K\}$ and applies a Gumbel–Softmax layer (Jang et al., 2017) to obtain a differentiable code vector $\tilde{C}$ during training; at inference, we take the arg-max to obtain discrete codes. The *context branch* produces an embedding $\psi(X)$, which we concatenate with $\tilde{C}$ and feed into an outcome predictor f_θ trained with MSE. For the overlap distance $\widehat{D}$, we use the sliced Wasserstein distance (Kolouri et al., 2019) on minibatches of $\psi(X)$, comparing the pooled distribution to each within-code distribution $\psi(X) \mid \tilde{C} = c$; we stop gradients through ψ in this term, so that the overlap penalty adjusts code assignments via ϕ rather than by distorting the context representation. Because the overlap term is minimized trivially by collapsing all actions into a single code, we add a marginal code-balance penalty $\gamma\,\mathrm{KL}(\widehat{p}_C \,\|\, \mathrm{Unif})$, where $\widehat{p}_C$ is the empirical code histogram in the minibatch. The total Stage 1 loss is thus $\mathcal{L}_{\mathrm{pred}} + \beta\mathcal{L}_{\mathrm{ov}} + \gamma\mathcal{L}_{\mathrm{cb}}$.

Stage ②: Learning a code-level policy. To fit the code-level regression $\widehat{m}(x, c)$, we initialize the predictor f_θ from Stage 1 and fine-tune the prediction head on (X_i, C_i, Y_i) with the encoder frozen, reusing the Stage 1 representation while adapting the head to hard codes. We then define the code-level policy by $\pi_C(x) \in \arg\max_{c:\, \widehat{\mu}_C(c|x) \ge \tau} \widehat{m}(x, c)$, where $\widehat{\mu}_C(c \mid x)$ is a propensity proxy

[3] Standard choices include Wasserstein-1 (under Lipschitzness of e_c^2) or kernel MMD (under RKHS control of e_c^2) (Johansson et al., 2022).

fit by cross-entropy on (X_i, C_i). Using the estimated propensity, this approximates the feasible-class constraint $\Pi_{C,\tau}$ while optimizing the regression-adjusted objective in Eq. (3).

Stage ③: Deployment. For deployment at context x, we select the target code greedily as $c^\star = \pi_C(x)$. We then run the acceptance loop of Algorithm 1 with budget $T_{\max}$. Since the policy optimization already restricts to supported codes satisfying $\widehat{\mu}_C(c^\star \mid x) \geq \tau$, no additional overlap fallback is required at deployment. The cap $T_{\max}$ still controls worst-case deployment cost and limits deviation from the behavior policy when matching target codes is difficult.

7 Experiments

We evaluate PolicyLift in three steps. ① We use synthetic experiments to test the theoretical properties of behavior-consistent lifting, outcome-predictive code learning, and overlap-aware code learning. ② To show the modularity of PolicyLift, we evaluate the policy improvement across three datasets while varying the Stage ① code and Stage ② policy learning objective. ③ We assess practical applicability on a challenging, realistic, public language-generation benchmark and compare against prompt-based methods with additional information.

Datasets. For evaluation, we use three datasets that progress from isolated mechanisms to fully realistic generated text. (i) Our *synthetic* DGP uses numeric structured actions and separately controls within-code reward variation, code predictiveness, and overlap (Appendix D). (ii) We construct a *headline generation* benchmark that combines real article contexts with GPT-5.4 mini-generated headline actions and a controlled reward function. We use MiniLM embeddings with constructed simulator features (Appendices E and H.1) to ensure that a learnable signal can be retrieved from the noisy data. (iii) We test on the public *OfflinePrompts* (Kiyohara et al., 2025) benchmark that provides a MovieLens-based personalization benchmark with Mistral-generated text and a public reward simulator trained on real-world user responses. Here, PolicyLift learns from context, generated text, and reward and does **not** require observing the generating prompt like other methods (Appendix F).

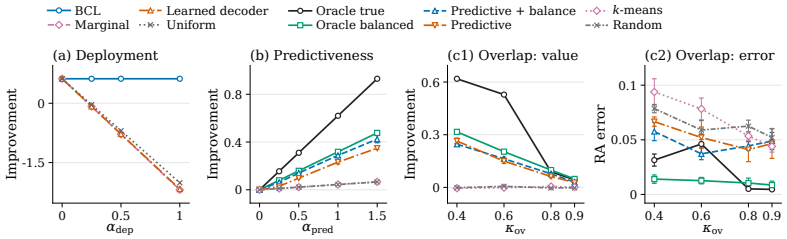


Figure 2: **Synthetic experiments.** (a) Deployment; (b) predictiveness; (c1–c2) overlap. (a,b,c1) Policy improvement (higher better); (c2) RA estimation error (lower better). 5 seeds, std errors where available. Each diagnostic fixes other stages; see Appendix D.4.

Baseline. Related methods (Saito & Joachims, 2022; Saito et al., 2023; 2025) are *not directly applicable* due to important differences; these works (a) have mostly used a large but finite action space, which prevents generating new actions (e.g., the within-cluster policy of PoTEC (Saito et al., 2025)); or (b) make additional assumptions such as access to an observed discrete action cluster (like discrete prompt strategies (Kiyohara et al., 2025)) or are OPE methods rather than OPL (Saito & Joachims, 2022; Saito et al., 2023). However, to demonstrate the modularity of our framework, we show we can integrate the aforementioned methods into PolicyLift for extending them to *generative policy learning*. We proceed as follows: For Stage 1, we replace the codes learned with our full objective with random codes, k-means codes, or predictive-only codes ($\beta = \gamma = 0$), while keeping RA and behavior-consistent lifting. For Stage 2, we retain the proposed codes but use code-IPS, code-DR, or kernel-IS objectives to directly adapt the methods from OPE MIPS (Saito & Joachims, 2022), OPE OffCEM (Saito et al., 2023), and sentence-level importance weighting (Kiyohara et al., 2025). On OfflinePrompts (where *additionally* prompt strategies are observed), we also compare with prompt-based regression and DSO (Kiyohara et al., 2025). These baseline can use prompt information that is *unavailable* to PolicyLift. Appendix H.1 describes the implementations. Our primary metric is policy improvement over behavior, $V(\hat{\pi}^\uparrow) - V(\mu)$. Tables report mean $\pm$ standard error over five seeds.

① **Validation of theoretical properties.** Figure 2 isolates the following three mechanisms (see Appendix D.4 for details): **(i)** To compare BCL against alternative decoding (uniform, context-free,

and learned decoding), we vary within-code reward effects with oracle codes fixed. **(ii)** To isolate the effect of the code quality, we vary the predictive signal with oracle Stage 2 optimization. **(iii)** We vary logging-policy concentration to examine overlap and estimation error. **Findings: (i)** BCL preserves code-policy value and guarantees improvement while alternative decoders show the deployment gap, i.e., even *value deterioration* (Figure 2, (a)). **(ii)** Outcome-predictive codes increase the value, i.e., enabling larger policy improvement (Figure 2, (b)). The overlap diagnostic demonstrates the tradeoff between representation quality (predictiveness) and reliable policy learning (stage 2 error) (Figure 2 (c)), showing the code value in theory for improvement (c1), and how consistently we can learn a good policy on these codes from finite data (c2). ⇒ *The experiments demonstrate the importance of our three main modeling choices: predictive codes, support-constrained policy learning, and behavior-consistent lifting.*

② **Modularity and policy improvement.** Table 2 shows results for various configurations of POLICYLIFT. Importantly, we show positive mean improvement for *every* configuration of POLICYLIFT on all three datasets. Hence, POLICYLIFT is flexible and can be combined with various stage 1 and 2 alternatives to extend them for generative policy improvement. Between the configurations, we observe the following: ① **Code learning:** Our proposed code learning improves over the behavior policy and achieves higher mean improvement than methods without predictiveness (k-means, random) across all datasets. Additional balancing improves on the synthetic and semi-synthetic datasets with challenging overlap, with an increase in improvement on the headline data from 0.0059 to 0.0482, and stays competitive for OfflinePrompts. ② **Policy learning:** With our proposed codes fixed, our proposed RA objective achieves the highest improvement on synthetic data and OfflinePrompts, and remains competitive with the alternative objectives on headlines. ⇒ *Our proposed configuration is a very strong default, and our* POLICYLIFT *also can extend alternative estimators to produce effective generative policies.*

Table 2: **POLICYLIFT ablations.** Policy improvement (↑); cells report mean ± SE over 5 seeds. Green: proposed configuration. Red: mean ± SE intervals entirely below proposed. Gray: overlapping intervals.

Variant	Synthetic	Headlines	Offline Prompts
Proposed:			
Pred.&bal. codes + Code-RA	0.1579 ± 0.0083	0.0482 ± 0.0095	0.1332 ± 0.0034
Stage 1: code replacement			
Random	0.0049 ± 0.0040	0.0400 ± 0.0084	0.0116 ± 0.0019
k-means	0.0015 ± 0.0040	0.0219 ± 0.0000	0.0902 ± 0.0036
Predictive only	0.1547 ± 0.0076	0.0059 ± 0.0026	0.1498 ± 0.0280
Stage 2: objective replacement			
Code-IPS (Saito & Joachims, 2022)	0.1437 ± 0.0115	0.0522 ± 0.0058	0.1031 ± 0.0053
Code-DR (Saito et al., 2023)	0.1334 ± 0.0257	0.0529 ± 0.0063	0.1088 ± 0.0030
Kernel-IS (Kiyohara et al., 2025)	0.1083 ± 0.0229	0.0448 ± 0.0095	0.1112 ± 0.0086

③ **Policy improvement without logged prompts.** OfflinePrompts tests whether the framework remains effective on fully realistic generated text even beyond our carefully designed headline benchmark. Table 3 compares the same proposed configuration against methods that evaluate and select different prompts that generated the text explicitly; hence, POLICYLIFT uses *strictly less information* than the baselines. POLICYLIFT increases the policy value from 0.1031 to 0.2363, a relative improvement of approximately 129%. Its mean exceeds prompt-based regression, and compares similarly to DSO; however, DSO suffers from higher variation across seeds. ⇒ POLICYLIFT *demonstrates clear policy improvement and strong performance also in challenging realistic settings and even despite access to less information.*

Table 3: **OfflinePrompts.** Policy value (↑), mean ± SE over five seeds. Green: POLICYLIFT. Red: behavior. Gray: references using additional prompt information unavailable to POLICYLIFT; their mean ± SE intervals overlap its interval.

Method	Policy value
Behavior policy	0.1031 ± 0.0000
POLICYLIFT	0.2363 ± 0.0034
Information-advantaged references	
Prompt-based regression	0.2266 ± 0.0109
DSO-style prompt policy	0.2684 ± 0.0458

Discussion. We prove theoretically and show empirically that POLICYLIFT improves policies with structured actions by learning in a supported code space and deploying through behavior-consistent lifting, which removes the deployment gap. Our framework has two main limitations: it requires sampling access to the behavior system at deployment time, and preserves the within-code behavior distribution but does not optimize among actions with the same code. Future work could study less conservative lifting rules, further improved code representations, and online deployed evaluations of rolled-out policies beyond public benchmarking, such as A/B testing in personalized online marketing. To avoid negative impacts, we emphasize that deployment must be done in a safe, ethical, and reliable manner.

AI USE STATEMENT

The research questions and core methodological ideas were developed by the authors. Generative AI tools were used as research assistants for literature retrieval, feedback on research and experimental design, implementation and debugging, synthetic-data and figure preparation, checking and refining mathematical claims and proofs, interpreting results, and drafting and editing parts of the manuscript. All AI-assisted outputs were reviewed and, where applicable, verified against the implementation, experiments, literature, or mathematical derivations; the authors made all final decisions and take responsibility for the content. LLM use as part of the experimental data-generation pipelines is described separately in the corresponding experimental appendices.

REPRODUCIBILITY STATEMENT

Assumptions and proofs are provided in Appendix C. Dataset construction and evaluation protocols are described in Appendices D, E, and F. Appendix H provides implementation details, experimental configurations, and the anonymous code link.

ETHICS STATEMENT

Our experiments use existing datasets, generated text, and simulator-based rewards rather than deploying policies to users. Potential applications of generative policy improvement to personalized communication should consider privacy, fairness, factual accuracy, safety, and user well-being alongside the optimized outcome.

REFERENCES

Susan Athey and Stefan Wager. Policy learning with observational data. *Econometrica*, 89(1): 133–161, 2021.

Jiayu Chen, Bhargav Ganguly, Yang Xu, Yongsheng Mei, Tian Lan, and Vaneet Aggarwal. Deep generative models for offline policy learning: Tutorial, survey, and perspectives on future directions. *Transactions on Machine Learning Research*, 2024.

Matej Cief, Jacek Golebiowski, Philipp Schmidt, Ziawasch Abedjan, and Artur Bekasov. Learning action embeddings for off-policy evaluation. In *European Conference on Information Retrieval*, pp. 108–122. Springer, 2024.

Richard K Crump, V Joseph Hotz, Guido W Imbens, and Oscar A Mitnik. Dealing with limited overlap in estimation of average treatment effects. *Biometrika*, 96(1):187–199, 2009.

Mert Demirer, Vasilis Syrgkanis, Greg Lewis, and Victor Chernozhukov. Semi-Parametric Efficient Policy Learning with Continuous Actions. In *NeurIPS*, 2019. URL https://arxiv.org/pdf/1905.10116.

Miroslav Dudík, Dumitru Erhan, John Langford, and Lihong Li. Doubly Robust Policy Evaluation and Optimization. *Statistical Science*, 29(4), 2014. ISSN 0883-4237. doi: 10.1214/14-STS500. URL https://arxiv.org/pdf/1503.02834.

Alexander D'Amour, Peng Ding, Avi Feller, Lihua Lei, and Jasjeet Sekhon. Overlap in observational studies with high-dimensional covariates. *Journal of Econometrics*, 221(2):644–654, 2021.

Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. More robust doubly robust off-policy evaluation. In *International Conference on Machine Learning*, pp. 1447–1456. PMLR, 2018.

Nicolò Felicioni, Maurizio Ferrari Dacrema, Marcello Restelli, and Paolo Cremonesi. Off-policy evaluation with deficient support using side information. *Advances in Neural Information Processing Systems*, 35:30250–30264, 2022.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, 2019.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pp. 10835–10866. PMLR, 2023.

Lin Gui, Cristina Gârbacea, and Victor Veitch. Bonbon alignment for large language models and the sweetness of best-of-n sampling. *Advances in Neural Information Processing Systems*, 37: 2851–2885, 2024.

Shonosuke Harada and Hisashi Kashima. Graphite: Estimating individual effects of graph-structured treatments. In *Proceedings of the 30th ACM international conference on information & knowledge management*, pp. 659–668, 2021.

Audrey Huang, Adam Block, Qinghua Liu, Nan Jiang, Akshay Krishnamurthy, and Dylan J Foster. Is best-of-n the best of them? coverage, scaling, and optimality in inference-time alignment. In *International Conference on Machine Learning*, pp. 25075–25126. PMLR, 2025.

Guido W Imbens and Donald B Rubin. *Causal inference in statistics, social, and biomedical sciences*. Cambridge university press, 2015.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

Ying Jin, Zhimei Ren, Zhuoran Yang, and Zhaoran Wang. Policy learning "without" overlap: Pessimism and generalized empirical bernstein's inequality. *The Annals of Statistics*, 53(4): 1483–1512, 2025.

Yuu Jinnai, Tetsuro Morimura, Kaito Ariu, and Kenshi Abe. Regularized best-of-n sampling with minimum bayes risk objective for language model alignment. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 9321–9347, 2025.

Fredrik Johansson, Uri Shalit, and David Sontag. Learning representations for counterfactual inference. In *International conference on machine learning*, pp. 3020–3029. PMLR, 2016.

Fredrik D Johansson, Uri Shalit, Nathan Kallus, and David Sontag. Generalization bounds and representation learning for estimation of potential outcomes and causal effects. *Journal of Machine Learning Research*, 23(166):1–50, 2022.

Nathan Kallus. More efficient policy learning via optimal retargeting. *Journal of the American Statistical Association*, 116(534):646–658, 2021. doi: 10.1080/01621459.2020.1788948.

Nathan Kallus and Angela Zhou. Policy evaluation and optimization with continuous treatments. In *AISTATS*, 2018.

Nathan Kallus, Xiaojie Mao, Kaiwen Wang, and Zhengyuan Zhou. Doubly robust distributionally robust off-policy evaluation and learning. *arXiv preprint*, 2022. URL `http://arxiv.org/pdf/2202.09667v1`.

Toru Kitagawa and Aleksey Tetenov. Who should be treated? empirical welfare maximization methods for treatment choice. *Econometrica*, 86(2):591–616, 2018.

Haruka Kiyohara, Daniel Yiming Cao, Yuta Saito, and Thorsten Joachims. An off-policy learning approach for steering sentence generation towards personalization. In *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, pp. 41–50, 2025.

Soheil Kolouri, Kimia Nadjahi, Umut Simsekli, Roland Badeau, and Gustavo Rohde. Generalized sliced wasserstein distances. *Advances in neural information processing systems*, 32, 2019.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-learning for offline reinforcement learning. In *NeurIPS*, 2020.

J Nathan Matias, Kevin Munger, Marianne Aubin Le Quere, and Charles Ebersole. The upworthy research archive, a time series of 32,487 experiments in us media. *Scientific Data*, 8(1):195, 2021.

Elyas Meguellati, Lei Han, Abraham Bernstein, Shazia Sadiq, and Gianluca Demartini. How good are llms in generating personalized advertisements? In *Companion Proceedings of the ACM Web Conference 2024*, pp. 826–829, 2024.

Sidharth Mudgal, Jong Lee, Harish Ganapathy, Yaguang Li, Tao Wang, Yanping Huang, Zhifeng Chen, Heng-Tze Cheng, Michael Collins, Trevor Strohman, et al. Controlled decoding from language models. In *International Conference on Machine Learning*, pp. 36486–36503. PMLR, 2024.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Jie Peng, Hao Zou, Jiashuo Liu, Shaoming Li, Yibao Jiang, Jian Pei, and Peng Cui. Offline policy evaluation in large action spaces via outcome-oriented action grouping. In *Proceedings of the ACM Web Conference 2023*, pp. 1220–1230, 2023.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Paul R Rosenbaum and Donald B Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983.

Donald B. Rubin. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66(5):688–701, 1974. ISSN 0022-0663. doi: 10.1037/h0037350.

Noveen Sachdeva, Yi Su, and Thorsten Joachims. Off-policy bandits with deficient support. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 965–975, 2020.

Noveen Sachdeva, Lequn Wang, Dawen Liang, Nathan Kallus, and Julian McAuley. Off-policy evaluation for large action spaces via policy convolution. In *Proceedings of the ACM Web Conference 2024*, pp. 3576–3585, 2024.

Yuta Saito and Thorsten Joachims. Off-policy evaluation for large action spaces via embeddings. In *International Conference on Machine Learning (ICML)*, 2022.

Yuta Saito, Qingyang Ren, and Thorsten Joachims. Off-policy evaluation for large action spaces via conjunct effect modeling. In *international conference on Machine learning*, pp. 29734–29759. PMLR, 2023.

Yuta Saito, Jihan Yao, and Thorsten Joachims. Potec: Off-policy contextual bandits for large action spaces via policy decomposition. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jonas Schweisthal, Dennis Frauen, Valentyn Melnychuk, and Stefan Feuerriegel. Reliable off-policy learning for dosage combinations. *Advances in Neural Information Processing Systems*, 36: 67900–67924, 2023.

Uri Shalit, Fredrik D Johansson, and David Sontag. Estimating individual treatment effect: generalization bounds and algorithms. In *International conference on machine learning*, pp. 3076–3085. PMLR, 2017.

Ashish Sharma, Inna W Lin, Adam S Miner, David C Atkins, and Tim Althoff. Human–ai collaboration enables more empathic conversations in text-based peer-to-peer mental health support. *Nature Machine Intelligence*, 5(1):46–57, 2023.

Lei Shi, David Arbour, Raghavendra Addanki, Ritwik Sinha, and Avi Feller. Leveraging semantic similarity for experimentation with ai-generated treatments. *Advances in Neural Information Processing Systems*, 2025.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.

Adith Swaminathan and Thorsten Joachims. Counterfactual risk minimization: learning from logged bandit feedback. In *ICML*, 2015. URL `https://arxiv.org/pdf/1502.02362`.

Cameron Voloshin, Hoang M Le, Nan Jiang, and Yisong Yue. Empirical study of off-policy policy evaluation for reinforcement learning. *Advances in Neural Information Processing Systems*, 2021.

Kevin Yang and Dan Klein. Fudge: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3511–3535, 2021.

Pan Zhao, Antoine Chambaz, Julie Josse, and Shu Yang. Positivity-free policy learning with observational data. In *International Conference on Artificial Intelligence and Statistics*, pp. 1918–1926. PMLR, 2024.

Wenxuan Zhou, Sujay Bajracharya, and David Held. Plas: Latent action space for offline reinforcement learning. In *Conference on Robot Learning*, pp. 1719–1735. PMLR, 2021.

A EXTENDED DISCUSSION AND LIMITATIONS

POLICYLIFT addresses policy improvement with structured actions by learning a code-level policy and deploying it through behavior-consistent lifting. This separates policy learning from action generation: the policy is optimized in a supported code space, while deployment remains in the original action space by sampling from the behavior system. We show theoretically and empirically that this removes the deployment gap for behavior-consistent lifting and transfers code-level improvement to deployed improvement over the behavior policy.

The approach has different limitations. First, POLICYLIFT requires sampling access to the behavior system at deployment time. It does not apply when the logging policy cannot be queried or when deployment must use a different generator. Second, behavior-consistent lifting preserves the within-code behavior distribution. This gives an identified deployment rule, but it does not optimize among actions with the same code. If reliable within-code outcome scores or model-weight access are available, less conservative instantiation rules may improve further, but they require additional assumptions to avoid a deployment gap. Third, the method depends on learning codes that are both predictive of outcomes and sufficiently supported. If no code representation satisfies this predictiveness–overlap trade-off (e.g. due to a close-to-deterministic behavior system), the attainable improvement is limited. All three represent interesting directions for future work, as does evaluating our framework in real-world settings, e.g., via A/B testing in personalized online marketing. In such applications, our framework could improve personalization and decision-making in generative systems, though care should be taken to ensure that optimization objectives remain aligned with user well-being and broader platform goals.

B Extended Related Work

LLMs and generative alignment. Several adjacent literatures improve policies that can produce or select complex outputs, but rely on different information or deployment assumptions. RLHF and direct preference optimization fine-tune generative models from preference data and require access to model weights (Ouyang et al., 2022; Rafailov et al., 2023). Controlled decoding steers a frozen generator at inference time, but requires decoding-time access and an external reward, value, or attribute scorer (Mudgal et al., 2024; Yang & Klein, 2021). Inference-time reranking and Best-of-N sampling select among generated candidates using a learned reward model, typically trained from preference data (Stiennon et al., 2020; Nakano et al., 2021; Gao et al., 2023). Plain Best-of-N can overoptimize such proxy rewards as the candidate set grows (Gao et al., 2023); recent regularized variants mitigate this effect, but still optimize in the original structured action space using reward-model scores (Gui et al., 2024; Huang et al., 2025; Jinnai et al., 2025). In contrast, PolicyLift starts only from logged observational triples (X, A, Y): it does not assume preference data, reward-model supervision, model-weight access, action-level likelihoods, or a scorer that can evaluate arbitrary generated candidates.

Offline reinforcement learning. Offline RL learns from logged data using support constraints, pessimism, or behavior regularization (Fujimoto et al., 2019; Kumar et al., 2020; Kostrikov et al., 2022), but typically operates directly in the original action space rather than learning a deployable representation-level policy. In a similar setting, deep generative policy-learning methods use generative models and latent action decoders for policies (Chen et al., 2024; Zhou et al., 2021), but require training such models and do not rule out a deployment gap when latent actions are decoded into original actions.

Balanced representation learning in causal inference. Related work on balanced representations for treatment effect estimation uses distributional distance regularization on covariate representations to adjust for covariate shift between treatment groups (Johansson et al., 2016; Shalit et al., 2017; Johansson et al., 2022), but assumes discrete treatments. Harada & Kashima (2021) extend this balancing to high-dimensional treatments. However, these works consider only treatment effect estimation and not policy learning. Relatedly, Shi et al. (2025) learn representations of high-dimensional treatments for causal effect estimation and adaptive online experimentation, while our focus is offline policy learning from fixed observational data.

Table 4: **Differences to selected related streams.** We compare PolicyLift to the most relevant lines of work along the two challenges emphasized in this paper: overlap in structured action spaces and deployment of a representation-level policy as a policy over original structured actions.

	Obs. outcomes	Structured actions	Learned repr.	Overlap controlled	Deploys in orig. space
Standard OPE/OPL (Dudík et al., 2014; Swaminathan & Joachims, 2015; Kallus, 2021)	✓	✗	–	(✗)	–
Action embeddings / clusters (Saito & Joachims, 2022; Saito et al., 2023; 2025)	✓	✓	(✓)	(✗)	✗
Learned action representations (Peng et al., 2023; Cief et al., 2024)	✓	✓	✓	(✗)	✗
Prompt-guided generation (Kiyohara et al., 2025)	✓	✓	✗	(✗)	(✓)
Decoder-based offline RL (Fujimoto et al., 2019; Zhou et al., 2021)	✓	(✓)	✓	(✓)	(✓)
Inference-time reranking / Best-of-N (Gao et al., 2023; Gui et al., 2024; Huang et al., 2025)	✗	✓	✗	–	✓
PolicyLift (ours)	✓	✓	✓	✓	✓

Legend. ✓ = addressed; ✗ = not addressed; (✓) = partially addressed; (✗) = assumed or not enforced; – = not applicable. "Overlap controlled" means that overlap is encouraged or enforced by the learning procedure rather than only assumed for identification. "Deploys in original space" means that the learned policy is deployable as a policy over original structured actions.

Research gap. Existing methods address only parts of our setting: overlap-constrained OPL targets simpler action spaces, representation-based OPE/OPL typically assumes fixed actions and

representation-level overlap, and generative alignment or offline RL requires stronger access such as preference data, reward models, model weights, or learned generators. This leaves open generative policy improvement from observational data. POLICYLIFT addresses this gap by learning overlap-aware action codes, optimizing a code-level policy, and lifting the policy through the behavior system.

C Proofs and Extensions

C.1 A technical assumption on within-code conditionals

For the results below, we assume existence of regular conditional distributions for the within-code behavior.

Assumption A3 (Within-code behavior conditional). For each $x \in \mathcal{X}$ and $c \in \{1, \dots, K\}$ with $\mu_C(c \mid x) > 0$, there exists a conditional distribution $\mu(\cdot \mid x, c)$ such that

$$\mathbb{P}(A \in S \mid X = x, C = c) = \int_S \mu(a \mid x, c)\, \mathrm{d}a \tag{11}$$

for all measurable $S \subseteq \mathcal{A}$ (interpreting $\mathrm{d}a$ generically; in discrete $\mathcal{A}$ replace by sums).

C.2 Deployment gap and context-free decoding

Proof of Theorem 5.1. Let $\tilde{A} \sim \pi^\kappa(\cdot \mid X)$. By consistency and ignorability,

$$V(\pi^\kappa) = \mathbb{E}_X\Big[\mathbb{E}[Y(\tilde{A}) \mid X]\Big] = \mathbb{E}_X\left[\sum_{a \in \mathcal{A}} \pi^\kappa(a \mid X)\, m(X, a)\right] \tag{12}$$

$$= \mathbb{E}_X\left[\sum_{c=1}^{K} \pi_C(c \mid X) \sum_{a \in \mathcal{A}} \kappa(a \mid X, c)\, m(X, a)\right] \tag{13}$$

$$= \mathbb{E}_X\left[\sum_{c=1}^{K} \pi_C(c \mid X)\, \mathbb{E}_{A \sim \kappa(\cdot \mid X, c)}[m(X, A)]\right]. \tag{14}$$

Subtracting and adding $m_C(X, c) = \mathbb{E}_{A \sim \mu(\cdot \mid X, c)}[m(X, A)]$, which holds by Lemma C.3 below, yields

$$V(\pi^\kappa) = \mathbb{E}_X\left[\sum_{c=1}^{K} \pi_C(c \mid X)\, m_C(X, c)\right] + \mathbb{E}_X\left[\sum_{c=1}^{K} \pi_C(c \mid X)\, \Delta_\kappa(X, c)\right] \tag{15}$$

$$= V_C(\pi_C) + \mathbb{E}_X\left[\sum_{c=1}^{K} \pi_C(c \mid X)\, \Delta_\kappa(X, c)\right], \tag{16}$$

which is (5). The two claims on when $\Delta_\kappa(x, c) = 0$ are immediate from the definition of Δ_κ. □

Corollary C.1 (Context-free decoding can yield a nonzero deployment gap under perfect marginal fit)**.** *Suppose the decoder is context-free, $\kappa(\cdot \mid x, c) \equiv \kappa_0(\cdot \mid c)$, and matches the marginal within-code behavior distribution, $\kappa_0(\cdot \mid c) = \mu(\cdot \mid c) := \mathbb{P}(A \in \cdot \mid C = c)$. Then*

$$V(\pi^{\kappa_0}) - V_C(\pi_C) = \mathbb{E}_X\left[\sum_{c=1}^{K} \pi_C(c \mid X) \int_{\mathcal{A}} m(X, a)\, \big(\mu(\mathrm{d}a \mid c) - \mu(\mathrm{d}a \mid X, c)\big)\right]. \tag{17}$$

Consequently, perfect marginal fit does not guarantee a zero deployment gap.

Proof. Apply Theorem 5.1 with $\kappa(\cdot \mid x, c) = \mu(\cdot \mid c)$. The deployment gap at (x, c) simplifies to

$$\Delta_{\kappa_0}(x, c) = \mathbb{E}_{A \sim \mu(\cdot \mid c)}[m(x, A)] - \mathbb{E}_{A \sim \mu(\cdot \mid x, c)}[m(x, A)] \tag{18}$$

$$= \int_{\mathcal{A}} m(x, a)\, \big(\mu(\mathrm{d}a \mid c) - \mu(\mathrm{d}a \mid x, c)\big), \tag{19}$$

which gives (17) after substitution into (5). □

Corollary C.1 makes precise why context-free decoding can break alignment between code-level evaluation and deployed value: even perfect reconstruction of the *marginal* within-code distribution averages across contexts, whereas deployment requires the *context-specific* mixture $\mu(\cdot \mid x, c)$.

Proposition C.2 (Non-identifiability under support mismatch)**.** *Fix a context-code pair (x, c) with $\mu_C(c \mid x) > 0$. Suppose there exists an action $a^\star \in \mathcal{A}$ such that $g_\phi(a^\star) = c$ and*

$$\kappa(a^\star \mid x, c) > 0 \qquad \textit{but} \qquad \mu(a^\star \mid x, c) = 0. \tag{20}$$

Then $\mathbb{E}_{A\sim\kappa(\cdot|x,c)}[m(x, A)]$ is not identified from the observational distribution under consistency and ignorability. The policy value $V(\pi^\kappa)$ is likewise not identified if such support mismatch occurs on a set of context-code pairs with positive probability under $P_X(\mathrm{d}x)\pi_C(c \mid x)$.

Proof. Construct two data-generating processes that share the same behavior policy μ and agree on all potential outcomes $\{Y(a) : a \neq a^\star\}$, but assign different values to $\mathbb{E}[Y(a^\star) \mid X = x]$. Because $\mu(a^\star \mid x, c) = 0$, the action $a^\star$ is never observed under the behavior distribution at (x, c), so both processes induce the same observational distribution of (X, A, Y) and the same $m_C(x, c)$. However, since $\kappa(a^\star \mid x, c) > 0$, the quantity $\mathbb{E}_{A\sim\kappa(\cdot|x,c)}[m(x, A)]$ differs between the two processes. Thus, if such support mismatches occur on a set of positive $P_X(\mathrm{d}x)\pi_C(c \mid x)$-measure, $V(\pi^\kappa)$ is not identified either. □

C.3 Value equivalence and improvement transfer

We begin with a lemma showing that the code regression function admits a causal interpretation as a within-code mixture of potential-outcome regressions. This is the key identity used in the proof of Theorem 5.1 and the corollaries that follow.

Lemma C.3 (Code regression as within-code mixture)**.** *The code regression function $m_C(x, c) := \mathbb{E}[Y \mid X = x, C = c]$ is defined as a statistical object under the observational distribution of (X, C, Y). Under consistency and ignorability, and under Assumption A3, it admits the causal interpretation*

$$m_C(x, c) = \mathbb{E}_{A\sim\mu(\cdot|x,c)}[m(x, A)] \tag{21}$$

for any (x, c) with $\mu_C(c \mid x) > 0$.

Proof. By definition and consistency,

$$m_C(x, c) = \mathbb{E}[Y \mid X = x, C = c] = \mathbb{E}[Y(A) \mid X = x, C = c]. \tag{22}$$

Applying the tower property by conditioning on A and then using ignorability,

$$\begin{aligned}
\mathbb{E}[Y(A) \mid X = x, C = c] &= \mathbb{E}_{A|X=x,C=c}[\mathbb{E}[Y(A) \mid X = x, A, C = c]] && (23)\\
&= \mathbb{E}_{A|X=x,C=c}[\mathbb{E}[Y(A) \mid X = x, A]] && (24)\\
&= \mathbb{E}_{A|X=x,C=c}[m(x, A)] = \mathbb{E}_{A\sim\mu(\cdot|x,c)}[m(x, A)], && (25)
\end{aligned}$$

where the last equality uses Assumption A3. □

Proof of Corollary 5.2. Apply Theorem 5.1 with $\kappa(\cdot \mid x, c) = \mu(\cdot \mid x, c)$. Then $\Delta_\kappa(x, c) = 0$ for all (x, c) in the support of π_C, and (5) reduces to $V(\pi^\uparrow) = V_C(\pi_C)$. □

Proof of Corollary 5.3. Applying Corollary 5.2 with $\pi_C = \mu_C$ shows $V(\mu) = V_C(\mu_C)$, since $\mu(a \mid x) = \sum_{c=1}^K \mu_C(c \mid x)\,\mu(a \mid x, c)$. Combining with $V_C(\pi_C) \geq V_C(\mu_C)$ and Corollary 5.2 yields $V(\pi^\uparrow) = V_C(\pi_C) \geq V_C(\mu_C) = V(\mu)$. □

C.4 Acceptance sampling

Proposition C.4 (Acceptance sampling implements the behavioral decoder)**.** *Fix a context x and a target code $c^\star$ with $\mu_C(c^\star \mid x) > 0$. Define the behavioral decoder*

$$\kappa_\mu(S \mid x, c^\star) := \mathbb{P}_\mu(A \in S \mid X = x, g_\phi(A) = c^\star) = \frac{\int_S \mathbf{1}\{g_\phi(a) = c^\star\}\,\mu(\mathrm{d}a \mid x)}{\mu_C(c^\star \mid x)} \tag{26}$$

for all measurable $S \subseteq \mathcal{A}$. Let $A^{(1)}, A^{(2)}, \ldots \overset{\text{iid}}{\sim} \mu(\cdot \mid x)$ and define

$$T := \inf\{t \geq 1 : g_\phi(A^{(t)}) = c^\star\}, \qquad A^{\mathrm{RS}} := A^{(T)}. \tag{27}$$

Then, for all measurable $S \subseteq \mathcal{A}$,

$$\mathbb{P}(A^{\mathrm{RS}} \in S \mid X = x; c^\star) = \kappa_\mu(S \mid x, c^\star). \tag{28}$$

Proof. Let $p_c(x) := \mu_C(c^\star \mid x)$ and

$$q_S(x, c^\star) := \mathbb{P}_\mu(A \in S, g_\phi(A) = c^\star \mid X = x) = \int_S \mathbf{1}\{g_\phi(a) = c^\star\}\, \mu(\mathrm{d}a \mid x). \tag{29}$$

Since $p_c(x) > 0$, the stopping time T is finite almost surely. By independence of the proposal draws,

$$\mathbb{P}(A^{\mathrm{RS}} \in S \mid X = x; c^\star) = \sum_{t=1}^{\infty} \mathbb{P}(T = t, A^{(t)} \in S \mid X = x; c^\star) \tag{30}$$

$$= \sum_{t=1}^{\infty} (1 - p_c(x))^{t-1} q_S(x, c^\star) \tag{31}$$

$$= \frac{q_S(x, c^\star)}{p_c(x)} \tag{32}$$

$$= \frac{\int_S \mathbf{1}\{g_\phi(a) = c^\star\}\, \mu(\mathrm{d}a \mid x)}{\mu_C(c^\star \mid x)} = \kappa_\mu(S \mid x, c^\star), \tag{33}$$

where the last equality follows from (26). □

Fix a context x and a target code $c^\star$ with $\mu_C(c^\star \mid x) > 0$. The ideal acceptance sampler draws $A^{(1)}, A^{(2)}, \ldots \overset{\text{iid}}{\sim} \mu(\cdot \mid x)$ and returns $A^{(T)}$, where $T := \min\{t \geq 1 : g_\phi(A^{(t)}) = c^\star\}$ is the first accepted index.

Proof of Lemma 5.4. For any measurable $S \subseteq \mathcal{A}$,

$$\mathbb{P}(A \in S, g_\phi(A) = c^\star \mid X = x) = \int_S \mu(a \mid x)\, \mathbf{1}\{g_\phi(a) = c^\star\}\, \mathrm{d}a, \tag{34}$$

$$\mathbb{P}(g_\phi(A) = c^\star \mid X = x) = \mu_C(c^\star \mid x). \tag{35}$$

Conditioning on acceptance yields $\mathbb{P}(A \in S \mid X = x, g_\phi(A) = c^\star) = \mu(S \mid x, c^\star)$, so the returned action has distribution $\mu(\cdot \mid x, c^\star)$. The acceptance probability per draw is $\mu_C(c^\star \mid x)$, and T is geometric with mean $1/\mu_C(c^\star \mid x)$. On $\Pi_{C,\tau}$, the bound $\mu_C(c^\star \mid x) \geq \tau$ gives $\mathbb{E}[T \mid X = x, C = c^\star] \leq 1/\tau$. □

Finite-cap variant. In practice, the acceptance loop is capped at $T_{\max}$ draws, and we fall back to an independent draw from $\mu(\cdot \mid x)$. Let $p_{\mathrm{fail}}(x, c) := (1 - \mu_C(c \mid x))^{T_{\max}}$ be the probability of no acceptance within $T_{\max}$ draws. If $|Y| \leq B$ almost surely, then this fallback produces an $O(p_{\mathrm{fail}})$ deviation in deployed value:

$$\left|V(\pi^{\uparrow}_{T_{\max}}) - V(\pi^{\uparrow})\right| \;\leq\; 2B\, \mathbb{E}\left[\sum_{c=1}^{K} \pi_C(c \mid X)\, p_{\mathrm{fail}}(X, c)\right], \tag{36}$$

where $\pi^{\uparrow}_{T_{\max}}$ is the capped sampler's induced deployed policy. This quantifies the trade-off between computational cost (at most $T_{\max} + 1$ behavior draws) and approximation error (p_{fail} decaying exponentially in $T_{\max}$).

C.5 Stochastic encoder extension

The main text uses a deterministic encoder $C = g_\phi(A)$ to match the neural implementation at inference time. For completeness, we show that the value equivalence result extends to stochastic encoders.

Suppose the encoder is stochastic: $C \sim g_\phi(\cdot \mid A)$. The induced code-level behavior policy is $\mu_C(c \mid x) := \mathbb{P}(C = c \mid X = x)$, and the within-code behavior conditional is $\mu(a \mid x, c) := \mathbb{P}(A = a \mid X = x, C = c)$, both taken under the joint distribution over (A, C). Behavior-consistent lifting is defined exactly as in (4).

Theorem C.5 (Value equivalence with stochastic encoding)**.** *Under consistency and ignorability and under Assumption A3 applied to the joint (A, C), for any $\pi_C \in \Pi_{C,\tau}$ the behavior-consistent deployed policy satisfies $V(\pi^{\uparrow}) = V_C(\pi_C)$, where $m_C(x, c) := \mathbb{E}[Y \mid X = x, C = c]$ under the joint distribution.*

Proof. Identical to the proof of Corollary 5.2, using conditioning on (X, C) and Lemma C.3 for the joint distribution over (A, C). □

C.6 Proof of Theorem 5.5

Recall $\bar{m}(x) = \mathbb{E}[Y \mid X = x]$ and $m_C(x, c) = \mathbb{E}[Y \mid X = x, C = c]$.

Equivalent characterizations of $\mathrm{Sig}(\phi)$. By the tower property applied to C conditional on X,

$$\mathbb{E}_{C|X}[m_C(X, C)] = \mathbb{E}_{C|X}[\mathbb{E}[Y \mid X, C]] = \mathbb{E}[Y \mid X] = \bar{m}(X). \tag{37}$$

Hence

$$\mathrm{Sig}(\phi) = \mathbb{E}_X\left[\mathrm{Var}_{C|X}(m_C(X, C))\right] = \mathbb{E}_{X,C}\left[(m_C(X, C) - \bar{m}(X))^2\right], \tag{38}$$

which is the first equality in (7).

For the second equality, note that

$$\mathbb{E}[(Y - \bar{m}(X))^2] = \mathbb{E}_X[\mathrm{Var}(Y \mid X)], \qquad \mathbb{E}[(Y - m_C(X, C))^2] = \mathbb{E}_{X,C}[\mathrm{Var}(Y \mid X, C)]. \tag{39}$$

The law of total variance gives $\mathrm{Var}(Y \mid X) = \mathbb{E}_{C|X}[\mathrm{Var}(Y \mid X, C)] + \mathrm{Var}_{C|X}(\mathbb{E}[Y \mid X, C])$. Taking expectations over X and rearranging,

$$\mathbb{E}_X[\mathrm{Var}(Y \mid X)] - \mathbb{E}_{X,C}[\mathrm{Var}(Y \mid X, C)] = \mathbb{E}_X\left[\mathrm{Var}_{C|X}(\mathbb{E}[Y \mid X, C])\right] = \mathrm{Sig}(\phi), \tag{40}$$

which is the second equality in (7).

Oracle value lower bound. Fix x and let $Z := m_C(x, C)$ with $C \sim \mu_C(\cdot \mid x)$. Then $\mathbb{E}_{C|X=x}[Z] = \bar{m}(x)$ and

$$\mathrm{Var}(Z \mid X = x) = \mathbb{E}_{C|X=x}\left[(Z - \bar{m}(x))^2\right]. \tag{41}$$

Let

$$M_x := \max_c m_C(x, c), \qquad m_x := \min_c m_C(x, c). \tag{42}$$

Since $|Y| \le B$ almost surely, we have $m_C(x, c) \in [-B, B]$ for all (x, c), and hence $Z \in [m_x, M_x] \subseteq [-B, B]$.

We use the elementary bounded-support inequality

$$\mathrm{Var}(Z \mid X = x) \le \big(M_x - \mathbb{E}_{C|X=x}[Z]\big)\big(\mathbb{E}_{C|X=x}[Z] - m_x\big). \tag{43}$$

To see this, note that $(M_x - Z)(Z - m_x) \ge 0$ almost surely, so

$$Z^2 \le (M_x + m_x)Z - M_x m_x. \tag{44}$$

Taking expectations conditional on $X = x$ and subtracting $\mathbb{E}_{C|X=x}[Z]^2$ gives

$$\mathrm{Var}(Z \mid X = x) \le \big(M_x - \mathbb{E}_{C|X=x}[Z]\big)\big(\mathbb{E}_{C|X=x}[Z] - m_x\big). \tag{45}$$

Moreover, since $Z \in [-B, B]$,

$$\mathbb{E}_{C|X=x}[Z] - m_x \le 2B. \tag{46}$$

Therefore,

$$\mathrm{Var}(Z \mid X = x) \le 2B\big(M_x - \mathbb{E}_{C|X=x}[Z]\big). \tag{47}$$

Using $\mathbb{E}_{C|X=x}[Z] = \bar{m}(x)$ and rearranging yields

$$\max_c m_C(x, c) = M_x \ge \bar{m}(x) + \frac{1}{2B}\,\mathrm{Var}(Z \mid X = x). \tag{48}$$

Taking expectations over X and using $V_C^\star(\phi) = \mathbb{E}_X[\max_c m_C(X, c)]$ and $V(\mu) = \mathbb{E}_X[\bar{m}(X)]$ yields

$$V_C^\star(\phi) \ge V(\mu) + \frac{1}{2B}\,\mathbb{E}_X\left[\mathrm{Var}_{C|X}(m_C(X, C))\right] = V(\mu) + \frac{1}{2B}\,\mathrm{Sig}(\phi). \tag{49}$$

□

C.7 Proof of Theorem 5.6

Fix $\pi_C \in \Pi_C$ and define

$$\widehat{v}(x) := \sum_{c=1}^{K} \pi_C(c \mid x)\, \widehat{m}(x,c), \qquad v(x) := \sum_{c=1}^{K} \pi_C(c \mid x)\, m_C(x,c), \tag{50}$$

so that $\mathbb{E}_X[\widehat{V}_{\mathrm{RA}}(\pi_C)] = \mathbb{E}_X[\widehat{v}(X)]$ and $V_C(\pi_C) = \mathbb{E}_X[v(X)]$. Let $e_c(x) := m_C(x,c) - \widehat{m}(x,c)$ denote the code-level prediction residual.

Bias term. By linearity,

$$V_C(\pi_C) - \mathbb{E}_X[\widehat{V}_{\mathrm{RA}}(\pi_C)] = \mathbb{E}_X\left[\sum_{c=1}^{K} \pi_C(c \mid X)\, e_c(X)\right]. \tag{51}$$

Cauchy–Schwarz pointwise in X gives

$$\left|\sum_{c=1}^{K} \pi_C(c \mid X)\, e_c(X)\right| \le \sqrt{\sum_{c=1}^{K} \pi_C(c \mid X)\, e_c(X)^2}, \tag{52}$$

using $\sum_c \pi_C(c \mid X) = 1$. Taking expectations and applying Jensen's inequality (since $\sqrt{\cdot}$ is concave) yields

$$\left|V_C(\pi_C) - \mathbb{E}_X[\widehat{V}_{\mathrm{RA}}(\pi_C)]\right| \le \sqrt{\mathbb{E}_X\left[\sum_{c=1}^{K} \pi_C(c \mid X)\, e_c(X)^2\right]}. \tag{53}$$

Since $\pi_C(c \mid X) \le 1$ for all c,

$$\mathbb{E}_X\left[\sum_{c=1}^{K} \pi_C(c \mid X)\, e_c(X)^2\right] \le \sum_{c=1}^{K} \mathbb{E}_{P_X}[e_c(X)^2] \le \sum_{c=1}^{K} \mathbb{E}_{Q_c^{\phi}}[e_c(X)^2] + \lambda_{\mathrm{ov}} \sum_{c=1}^{K} D(P_X, Q_c^{\phi}), \tag{54}$$

where the last inequality is the shift bound (9). This gives the first term of (10).

Sample fluctuation term. Treating $\widehat{m}$ as fixed and independent of the evaluation sample (by cross-fitting), we have $|\widehat{v}(x)| \le B$ for all x, since $|\widehat{m}(x,c)| \le B$ and $\sum_c \pi_C(c \mid x) = 1$. Hoeffding's inequality and a union bound over the finite class Π_C imply that with probability at least $1 - \delta$,

$$\sup_{\pi_C \in \Pi_C} \left|\widehat{V}_{\mathrm{RA}}(\pi_C) - \mathbb{E}_X[\widehat{V}_{\mathrm{RA}}(\pi_C)]\right| \le B\sqrt{\frac{2\log(2|\Pi_C|/\delta)}{n}}. \tag{55}$$

Combining with the bias bound via the triangle inequality yields (10). □

EVALUATION BENCHMARKS

Evaluation design. Benchmarking off-policy learning against known ground-truth policy values requires counterfactual outcomes for actions that were not taken, which are unavailable in ordinary logged data. Controlled and semi-synthetic benchmarks are therefore standard for validating OPE/OPL methods against known ground truth, including in closely related large-action work using synthetic, supervised-to-bandit, and reward-simulator evaluations (Saito & Joachims, 2022; Saito et al., 2023; 2025; Kiyohara et al., 2025). We first evaluate POLICYLIFT on two deliberately controlled benchmarks with complementary purposes: the synthetic DGP isolates the mechanisms analyzed by the theory and enables oracle diagnostics, while the semi-synthetic headline benchmark combines real article contexts and LLM-generated structured actions with deliberately recoverable reward-relevant structure. Additionally, to evaluate the complete method in a less controlled, externally defined generative setting, we use the public OfflinePrompts benchmark with real MovieLens-derived contexts, raw generated text without injected simulator features, and a released reward simulator trained on real user ratings.

D SYNTHETIC DIAGNOSTICS

Here, we give the full data-generating process (DGP) and evaluation protocol for the synthetic diagnostics in Section 7. The goal is not to provide a broad benchmark, but to isolate the three mechanisms of POLICYLIFT under controlled conditions: behavior-consistent lifting (BCL), predictive action codes, and overlap-aware code learning. All diagnostics use the same DGP and vary only the relevant control parameter. The three parameters are: α_{dep}, which controls residual within-code action effects; α_{pred}, which controls the between-code outcome signal; and κ_{ov}, which controls code-level overlap.

D.1 DATA-GENERATING PROCESS

We generate logged triples (X_i, A_i, Y_i) from a simulator with hidden context type $Z_i^\star$, hidden simulator code $C_i^\star$, and structured numeric action A_i. The learner observes only (X_i, A_i, Y_i). The hidden variables are used only to construct oracle diagnostics.

Contexts. Let $K = 8$. For each unit,

$$Z^\star \sim \mathrm{Unif}\{0, \dots, K-1\}. \tag{56}$$

The observed context is

$$X = (e_{Z^\star} + \xi_x,\ u_x), \qquad \xi_x \sim \mathcal{N}(0, 0.05^2 I_K), \qquad u_x \sim \mathcal{N}(0, I_4), \tag{57}$$

where $e_z \in \mathbb{R}^K$ is the one-hot vector for context type z. Thus $X \in \mathbb{R}^{K+4}$. The hidden context type creates context-dependent optimal codes, while the low-dimensional observed context keeps oracle evaluation simple.

Logging policy over simulator codes. We define a relative code

$$D = (C^\star - Z^\star) \bmod K. \tag{58}$$

The logging policy samples D according to

$$p_{\kappa_{\mathrm{ov}}}(d) = \kappa_{\mathrm{ov}} \mathbf{1}\{d = 1\} + (1 - \kappa_{\mathrm{ov}}) \frac{1}{K}, \qquad d \in \{0, \dots, K-1\}, \tag{59}$$

and sets

$$C^\star = (Z^\star + D) \bmod K. \tag{60}$$

Equivalently, the true code propensity is

$$\mu_{C^\star}(c \mid z) = p_{\kappa_{\mathrm{ov}}}\big((c - z) \bmod K\big). \tag{61}$$

The relative-code construction makes the optimal code depend on the context while keeping all contexts symmetric. The parameter κ_{ov} controls overlap: larger values concentrate the logging policy on the plausible but suboptimal relative code $D = 1$.

Actions. The action vector has five blocks,

$$A = (A_{\text{abs}}, A_{\text{bal}}, A_{\text{dep}}, A_{\text{nuis}}, A_{\text{rand}}). \tag{62}$$

The first block carries a recoverable signal for the true simulator code:

$$A_{\text{abs}} = \lambda_{\text{abs}} e_{C^\star} + \eta_{\text{abs}}, \qquad \eta_{\text{abs}} \sim \mathcal{N}(0, \sigma^2_{\text{abs}} I_K). \tag{63}$$

Here $e_{C^\star}$ is the one-hot vector for the hidden code $C^\star$. This block makes the outcome-relevant code recoverable by predictive code learning.

The second block carries a signal for a more balanced auxiliary code. Define

$$P(D) = \begin{cases} 0, & D \in \{0,1\}, \\ 1, & D \in \{2,7\}, \\ 2, & D \in \{3,6\}, \\ 3, & D \in \{4,5\}, \end{cases} \qquad Q \sim \text{Bernoulli}(1/2), \qquad H^\star = 2P(D) + Q. \tag{64}$$

Then

$$A_{\text{bal}} = \lambda_{\text{bal}} e_{H^\star} + \eta_{\text{bal}}, \qquad \eta_{\text{bal}} \sim \mathcal{N}(0, \sigma^2_{\text{bal}} I_K). \tag{65}$$

The auxiliary code $H^\star$ creates a representation that is slightly coarser than $C^\star$ but better balanced across contexts, matching the predictiveness–overlap trade-off targeted by Stage 1. In our experiments, we use $H^\star$ as *oracle balanced codes*.

The third block creates residual within-code action variation:

$$A_{\text{dep}} = t_{Z^\star} + \eta_{\text{dep}}, \qquad \eta_{\text{dep}} \sim \mathcal{N}(0, \sigma^2_{\text{dep}} I_{d_{\text{dep}}}), \tag{66}$$

where $t_z \in \mathbb{R}^{d_{\text{dep}}}$ is a fixed context-specific prototype. This block makes the concrete action matter within a code and therefore tests whether deployment changes the within-code action distribution.

The fourth block is high-dimensional nuisance geometry:

$$A_{\text{nuis}} = \lambda_{\text{nuis}} q_J + \eta_{\text{nuis}}, \qquad J \sim \text{Unif}\{0, \dots, K-1\}, \qquad \eta_{\text{nuis}} \sim \mathcal{N}(0, \sigma^2_{\text{nuis}} I_{d_{\text{nuis}}}), \tag{67}$$

where $q_j \in \mathbb{R}^{d_{\text{nuis}}}$ are fixed nuisance prototypes. The nuisance block has no effect on the outcome but dominates Euclidean action geometry, so geometric clustering can fail.

Finally,

$$A_{\text{rand}} \sim \mathcal{N}(0, I_2). \tag{68}$$

This independent block is used only to construct random balanced codes.

Outcomes. Let

$$b(z) = 0.2 \sin(2\pi z / K) \tag{69}$$

and define the between-code reward signal by

$$s(z, c) = v_{(c-z) \bmod K}, \qquad (v_0, \dots, v_7) = (1.00, 0.65, 0.15, -0.15, -0.30, -0.10, 0.05, 0.30). \tag{70}$$

Thus the optimal relative code is $D = 0$, while the logging policy favors the suboptimal but plausible relative code $D = 1$.

The within-code residual effect is

$$r(z, A) = -\frac{1}{d_{\text{dep}}} \|A_{\text{dep}} - t_z\|_2^2. \tag{71}$$

Under behavior-consistent lifting,

$$\mathbb{E}[r(Z^\star, A) \mid Z^\star = z, C^\star = c] = -\sigma^2_{\text{dep}}, \tag{72}$$

which is constant in (z, c). Hence this term does not change the best code-level policy under behavior-consistent lifting; it only penalizes deployment rules that change the within-code action distribution.

The observed outcome is

$$Y = b(Z^\star) + \alpha_{\text{pred}} s(Z^\star, C^\star) + \alpha_{\text{dep}} r(Z^\star, A) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \sigma_y^2). \tag{73}$$

This separates the two relevant failure modes. The parameter α_{pred} controls the value of learning predictive codes, while α_{dep} controls the deployment loss from non-behavior-consistent instantiation.

D.2 ORACLE QUANTITIES

For the true simulator code $C^\star$, the code regression is available in closed form:

$$m_{C^\star}(z,c) = b(z) + \alpha_{\text{pred}} v_{(c-z) \bmod K} - \alpha_{\text{dep}} \sigma^2_{\text{dep}}. \tag{74}$$

The behavior value is

$$V(\mu) = \frac{1}{K} \sum_{z=0}^{K-1} \sum_{c=0}^{K-1} \mu_{C^\star}(c \mid z) m_{C^\star}(z,c). \tag{75}$$

The unconstrained oracle code value is

$$V^\star_{C^\star} = \frac{1}{K} \sum_{z=0}^{K-1} \max_{c \in \{0,\dots,K-1\}} m_{C^\star}(z,c). \tag{76}$$

For a support threshold τ, the support-constrained oracle value is

$$V^\star_{C^\star,\tau} = \frac{1}{K} \sum_{z=0}^{K-1} \max_{c : \mu_{C^\star}(c|z) \geq \tau} m_{C^\star}(z,c). \tag{77}$$

For a deterministic code policy $\hat{c}(z)$, the expected number of behavior-system draws required by behavior-consistent lifting is

$$\text{Draws}(\hat{c}) = \frac{1}{K} \sum_{z=0}^{K-1} \frac{1}{\mu_{C^\star}(\hat{c}(z) \mid z)}. \tag{78}$$

For learned or unsupervised representations $G = g(A)$, oracle quantities are estimated by a noise-free Monte Carlo sample from the known simulator. We estimate $\mu_G(g \mid z)$ and $m_G(z,g)$ by sample averages and then evaluate oracle policies by table lookup. This keeps the oracle diagnostics cheap and avoids training additional oracle models.

D.3 CODE REPRESENTATIONS

The diagnostics compare the following code representations.

Ground-truth oracle codes. The representation is $G = C^\star$. This is, in general, not available to the learner and serves as an upper benchmark for the information contained in the simulator code.

Oracle balanced codes. The representation is $G = H^\star$. This is, in general, not available to the learner and serves as an optimistic target performance for the learned balanced codes.

Random balanced codes. We construct G_{rand} by quantile-binning a fixed random projection of A_{rand} into K bins. These codes are balanced by construction but independent of the outcome-relevant action structure.

Geometric k-means codes. We fit k-means with K clusters on the raw action vector A. Since A_{nuis} is high-dimensional and high-variance, this representation tests whether purely geometric action clustering can be dominated by nuisance variation.

Learned predictive codes. We learn codes with the Stage 1 prediction objective and set the overlap and marginal-balance weights to zero. This representation tests whether outcome-predictive code learning can recover useful action structure from A.

Learned predictive and balanced codes. We learn codes with the full Stage 1 objective, including the prediction loss, distributional-overlap regularizer, and marginal code-balance regularizer. This representation tests whether overlap-aware code learning preserves most predictive signal while improving finite-sample reliability.

D.4 DIAGNOSTIC PROTOCOLS

Deployment diagnostic. The deployment diagnostic fixes the representation to $C^\star$, uses the oracle code-level policy, and uses the true support rule. It varies only the Stage 3 instantiation rule and α_{dep}. We compare behavior-consistent lifting, a marginal decoder $\mu(a \mid c)$, a learned Gaussian decoder $\hat{\kappa}(a \mid x, c)$, and a uniform within-code sampler. The main metric is deployment loss,

$$V_C(\pi_C^\star) - V(\pi_\kappa), \tag{79}$$

where $\pi_C^\star$ is the oracle code-level policy and π_κ is the deployed action-level policy induced by instantiation rule κ. This diagnostic isolates whether changing the within-code action distribution creates a gap between code-level value and deployed value.

Predictiveness diagnostic. The predictiveness diagnostic fixes behavior-consistent lifting and uses an oracle Stage 2 policy for each representation. It varies the code representation and α_{pred}. For a representation G, the oracle policy is

$$\pi_G^\star(z) \in \arg\max_g m_G(z, g). \tag{80}$$

We report oracle value, improvement over the behavior policy, representation regret relative to $C^\star$, and prediction error. Representation regret is

$$V_{C^\star}^\star - V_G(\pi_G^\star). \tag{81}$$

Because Stage 2 and Stage 3 are fixed to their oracle versions, differences in this diagnostic isolate the information contained in the learned or constructed code representation, and thus, show the change in the oracle attainable policy value.

Overlap diagnostic. The overlap diagnostic fixes behavior-consistent lifting and learns the Stage 2 policy with a regression-adjusted objective. It varies the representation, the support rule, and κ_{ov}. For each representation, we evaluate policies with and without the τ-constraint. The constrained policy selects only supported codes,

$$\hat{g}_\tau(z) \in \arg\max_{g:\hat{\mu}_G(g|z)\geq\tau} \hat{m}_G(z, g), \tag{82}$$

while the unconstrained policy maximizes $\hat{m}_G(z, g)$ over all codes. We report deployed value, improvement over the behavior policy, RA error, unsupported-code rate, and expected draws. The unsupported-code rate is

$$\mathbb{E}\left[\mathbf{1}\{\mu_G(\hat{g}(X) \mid X) < \tau\}\right], \tag{83}$$

and the RA error is

$$\left|\hat{V}_{\mathrm{RA}}(\hat{\pi}_G) - V_G(\hat{\pi}_G)\right|. \tag{84}$$

This diagnostic isolates whether overlap-aware code learning and the τ support rule improve reliability when code-level support becomes weak.

D.5 DEFAULT PARAMETERS

Unless otherwise stated, we use the parameters in Table 5. We average results over independent random seeds and report one standard error across seeds.

Table 5: Default parameters for the synthetic diagnostics.

Quantity	Value
Number of simulator codes and context types	$K = 8$
Observed context dimension	$K + 4 = 12$
Action block dimensions	$8 + 8 + 4 + 64 + 2$
Outcome noise	$\sigma_y = 0.25$
Absolute-code block	$\lambda_{\text{abs}} = 0.8,\ \sigma_{\text{abs}} = 0.15$
Balanced-code block	$\lambda_{\text{bal}} = 0.6,\ \sigma_{\text{bal}} = 0.25$
Deployment block	$d_{\text{dep}} = 4,\ \sigma_{\text{dep}} = 0.25$
Nuisance block	$d_{\text{nuis}} = 64,\ \lambda_{\text{nuis}} = 4.0,\ \sigma_{\text{nuis}} = 0.5$
Default predictiveness strength	$\alpha_{\text{pred}} = 1$
Default deployment-residual strength	$\alpha_{\text{dep}} = 0$
Default overlap severity	$\kappa_{\text{ov}} = 0.4$
Support threshold	$\tau = 0.03$
Deployment sweep	$\alpha_{\text{dep}} \in \{0, 0.25, 0.5, 1.0\}$
Predictiveness sweep	$\alpha_{\text{pred}} \in \{0, 0.25, 0.5, 1.0, 1.5\}$
Overlap sweep	$\kappa_{\text{ov}} \in \{0.4, 0.6, 0.8, 0.9\}$

E SEMI-SYNTHETIC BENCHMARK CONSTRUCTION

We construct a semi-synthetic text benchmark with real article excerpts as contexts X from the Upworthy Research Archive (Matias et al., 2021) (CC BY 4.0), LLM-generated headlines as actions A, and rewards assigned by a controlled simulator. The benchmark evaluates the full method in a noisy structured-text setting under limited code-level overlap while deliberately retaining recoverable reward-relevant structure. All methods receive the same numerical representations (Appendix E.4), which makes comparisons less sensitive to whether generic text embeddings capture the simulated reward-relevant structure. OfflinePrompts (Appendix F) separately evaluates raw generated text without injected simulator features.

E.1 STRUCTURAL DATA-GENERATING PROCESS

The simulator follows the graph

$$X \to Z^\star, \qquad Z^\star \to C^\star, \qquad (X, C^\star) \to A, \qquad (Z^\star, C^\star, A) \to Y.$$

For each unit i, we generate

$$\begin{aligned} Z_i^\star &= h_X(X_i), \\ C_i^\star &\sim \mu_{C^\star}(\cdot \mid Z_i^\star), \\ A_i &\sim G_{\text{acc}}(\cdot \mid X_i, C_i^\star), \\ Y_i &= m^\star(Z_i^\star, C_i^\star) + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}(0, \sigma^2). \end{aligned}$$

Learner inputs are specified in Appendix E.4; the annotators h_X and h_A are not provided to learning methods.

E.2 SIMULATOR CONTEXT AND ACTION SPACES

We use Upworthy article excerpts as contexts. The real data provide natural language inputs, but outcomes are assigned by the simulator rather than by observed click rates. This gives realistic text contexts while retaining known counterfactual rewards.

The hidden simulator context representation is

$$Z^\star = (T, U, N),$$

where T is the article topic, U is the tone, and N is the specificity need.

Table 6: Core notation for the semi-synthetic benchmark.

Symbol	Meaning
X	observed context text, here an Upworthy article excerpt
A	observed action text, here a generated headline
Y	observed scalar reward
$Z^\star \in \mathcal{Z}$	hidden simulator context representation
$C^\star \in \mathcal{C}^\star$	hidden simulator action code
$\mu_{C^\star}(c \mid z)$	code-level logging policy
$G(\cdot \mid X, C^\star)$	prompted headline generator
$G_{\mathrm{acc}}(\cdot \mid X, C^\star)$	accepted generator after validation
h_X	context annotator returning $Z^\star$
h_A	action validator returning realized code and faithfulness
$m^\star$	structural mean reward
$M_i(c)$	oracle code-level reward table for context i

Table 7: Hidden simulator context representation $Z^\star = (T, U, N)$.

Component	Meaning	Values
T	topic	`public_affairs`, `science_health`, `entertainment`, `sports_human`
U	tone	`informational`, `emotional`, `surprising`
N	specificity need	`low`, `high`

The hidden simulator action code is

$$C^\star = (S, R),$$

where S is the headline strategy and R is the tailoring level.

A `summary` headline states the core content directly. A `curiosity` headline withholds a key detail to induce curiosity without being misleading. An `emotion` headline foregrounds affective framing. A `direct` headline addresses the reader or frames the article as personally relevant. A `generic` headline avoids article-specific named entities or concrete details, while a `tailored` headline includes article-specific entities, details, or concrete context.

E.3 Reward-relevant strategy structure

For each topic–tone pair (T, U), we define a full simulator ranking of headline strategies

$$S_1^\star(T,U), S_2^\star(T,U), S_3^\star(T,U), S_4^\star(T,U),$$

where $S_1^\star(T,U)$ is best and $S_4^\star(T,U)$ is worst. Table 9 gives the ranking used in all experiments.

The optimal tailoring level is

$$R^\star(N) = \begin{cases} \texttt{generic}, & N = \texttt{low}, \\ \texttt{tailored}, & N = \texttt{high}. \end{cases}$$

Thus, the simulator-optimal code for context type $z = (T, U, N)$ is

$$c^\star(z) = \big(S_1^\star(T,U), R^\star(N)\big).$$

E.4 Structural reward model

For $z = (T, U, N)$ and $c = (s, r)$, the reward depends on the context-specific rank of the strategy s:

$$m^\star(z, c) = b(z) + \theta_{\mathrm{rank}(s;T,U)}. \tag{85}$$

Here $\mathrm{rank}(s; T, U) \in \{1, 2, 3, 4\}$ is given by the strategy lookup table in Table 9. The rank effects are

$$(\theta_1, \theta_2, \theta_3, \theta_4) = (1.00, 0.65, 0.15, -0.15). \tag{86}$$

Table 8: Hidden simulator action code $C^\star = (S, R)$.

Component	Meaning	Values
S	headline strategy	`summary`, `curiosity`, `emotion`, `direct`
R	tailoring level	`generic`, `tailored`

Table 9: Simulator ranking of headline strategies. Entries are ordered from best to worst.

Topic T	$U =$ `informational`	$U =$ `emotional`	$U =$ `surprising`
`public_affairs`	summary > direct > emotion > curiosity	direct > emotion > summary > curiosity	curiosity > direct > summary > emotion
`science_health`	summary > curiosity > direct > emotion	direct > emotion > summary > curiosity	curiosity > summary > direct > emotion
`entertainment`	direct > curiosity > emotion > summary	emotion > curiosity > direct > summary	curiosity > emotion > direct > summary
`sports_human`	summary > emotion > direct > curiosity	emotion > direct > summary > curiosity	curiosity > emotion > direct > summary

The additive context baseline is deterministic and does not affect the optimal action code:

$$b(z_i) = 0.2 \sin(2\pi\, r(z_i)/|\mathcal{Z}_{\text{obs}}|)\,, \tag{87}$$

where $r(z_i)$ is the index of the observed hidden context label among the labels in the constructed benchmark. In the reported benchmark there is no additional outcome noise, so

$$Y_i = m^\star(Z_i^\star, C_i^\star). \tag{88}$$

Policy values are evaluated using the same noise-free oracle reward table.

Observed numerical representations. The text learner operates on numerical representations $(\tilde{X}_i, \tilde{A}_i)$ that augment base text embeddings with controlled simulator-derived features: an exact context-label encoding and noisy action-code and strategy signals, alongside nuisance and random features. For contexts,

$$\tilde{X}_i = \big[\phi_X(X_i),\, 3.0\, e_{Z_i^\star}\big], \tag{89}$$

where $e_{Z_i^\star}$ is a one-hot encoding of the hidden context label. For actions,

$$\tilde{A}_i = \big[\phi_A(A_i),\, 0.8\, e_{C_i^\star} + \varepsilon_i^{(C)},\, 0.6\, e_{H_i^\star} + \varepsilon_i^{(H)},\, 4.0\, v_{J_i} + \varepsilon_i^{(J)},\, \varepsilon_i^{(R)}\big], \tag{90}$$

where $H_i^\star$ is the four-way strategy component of $C_i^\star$, v_{J_i} is a random normalized nuisance prototype, and the noise scales are 0.15, 0.25, and 0.5 for the $C^\star$, strategy, and nuisance blocks, respectively. The nuisance block has dimension 64 and the final random tail has dimension 2. This construction preserves the same generated texts, logging policy, and candidate pool, but makes explicit which reward-relevant and nuisance signals are available to representation learning.

E.5 Code-level logging policy

The logged simulator code is sampled from

$$\mu_{C^\star}(s, r \mid z) = \mu_S(s \mid T, U)\mu_R(r \mid N).$$

This factorization reflects that headline strategy depends on topic and tone, while tailoring depends on specificity need.

For strategy logging, we use an orientation parameter

$$d \in \{\text{optimal}, \text{suboptimal}\}$$

and a concentration parameter $\kappa_S \in [0, 1]$. Define

$$s^d_{\text{pref}}(T, U) = \begin{cases} S_1^\star(T, U), & d = \text{optimal}, \\ S_3^\star(T, U), & d = \text{suboptimal}. \end{cases}$$

Then

$$\mu_S(s \mid T, U) = \kappa_S \mathbf{1}\{s = s^d_{\text{pref}}(T, U)\} + (1 - \kappa_S)\frac{1}{|\mathcal{S}|}.$$

Thus, $\kappa_S = 0$ gives uniform strategy logging, while larger κ_S concentrates mass on the preferred logging strategy. In the suboptimal-oriented regime, the historical policy favors a plausible but non-optimal strategy while still assigning nonzero probability to the optimal one.

For tailoring, we use

$$\mu_R(r \mid N) = \kappa_R \mathbf{1}\{r = R^\star(N)\} + (1 - \kappa_R)\frac{1}{|\mathcal{R}|}.$$

Larger κ_R means that the logging policy more often chooses the tailoring level appropriate for the context.

With four strategies, $\kappa_S = 0.60$ assigns probability 0.70 to the preferred strategy and probability 0.10 to each other strategy. With two tailoring levels, $\kappa_R = 0.50$ assigns probability 0.75 to the appropriate tailoring level and probability 0.25 to the other level.

E.6 Prompt-based construction

The benchmark uses three prompt-based components: a context annotator h_X, a headline generator G, and an action validator h_A. All prompts return structured outputs.

Prompt 1: Context annotator h_X

Read the article excerpt and return JSON with keys `topic`, `tone`, and `specificity_need`. Use only the following values:

- `topic` $\in$ {`public_affairs`, `science_health`, `entertainment`, `sports_human`},
- `tone` $\in$ {`informational`, `emotional`, `surprising`},
- `specificity_need` $\in$ {`low`, `high`}.

Set `specificity_need = high` if an effective headline should mention article-specific entities, people, places, or concrete details; otherwise set it to `low`. Output only JSON.

Prompt 2: Headline generator G

Read the article excerpt. Write exactly one Upworthy-style headline that is faithful to the excerpt and does not invent facts.
Required strategy: `{summary | curiosity | emotion | direct}`.
Required tailoring level: `{generic | tailored}`.
Definitions:

- `summary`: state the core content of the article directly.
- `curiosity`: withhold a key detail to create curiosity without being misleading.
- `emotion`: foreground emotional or affective framing.
- `direct`: address the reader or make the article personally relevant.
- `generic`: avoid article-specific named entities or concrete details.
- `tailored`: include article-specific entities, details, or concrete context.

Keep the headline concise. If an observed headline is provided, use it only as a stylistic anchor and do not copy it verbatim. Output only the headline.

Prompt 3: Action validator h_A

Given an article excerpt and a headline, return JSON with keys `strategy`, `tailoring`, and `faithful`. Use only:

- `strategy` ∈ {`summary`, `curiosity`, `emotion`, `direct`},
- `tailoring` ∈ {`generic`, `tailored`},
- `faithful` ∈ {0,1}.

Set `faithful = 1` only if the headline is supported by the excerpt and does not introduce unsupported claims. Output only JSON.

E.7 Accepted action generator

The prompt $G(\cdot \mid X, C^\star)$ asks the language model to generate a headline with the requested simulator code. Since prompt compliance is not guaranteed, we validate the realized code using h_A. For intended code c, write

$$h_A(X, A) = (\tilde{C}^\star, \text{faithful}).$$

We accept a generated headline only if

$$\tilde{C}^\star = c \qquad \text{and} \qquad \text{faithful} = 1.$$

This defines the accepted generator

$$G_{\text{acc}}(\cdot \mid X, c) = G(\cdot \mid X, c,\ h_A(X, A) = (c, 1)).$$

Equivalently, G_{acc} is implemented by rejection sampling from G. This validation step is performed before assigning outcomes and does not use Y. Its role is to ensure that the observed text action A_i is semantically aligned with the simulator code $C_i^\star$ whose reward is assigned.

In implementation, we use a fixed retry budget $L_{\max}$. If no valid headline is obtained after $L_{\max}$ attempts, the unit is discarded.

E.8 Logged data and oracle evaluation

After construction, all learners use the representations specified in Appendix E.4 for the logged data

$$\mathcal{D}_{\log} = \{(X_i, A_i, Y_i)\}_{i=1}^n.$$

The simulator reward function and oracle counterfactual rewards are not provided to learners.

For the main benchmark, we store the oracle code-level reward table

$$M_i(c) = m^\star(Z_i^\star, c), \qquad c \in \mathcal{C}^\star.$$

The oracle code policy is

$$c_i^\star = \arg\max_{c \in \mathcal{C}^\star} M_i(c),$$

and its value is evaluated without outcome noise.

For any deployed text action $\tilde{A}_i$, we first validate the realized simulator code:

$$h_A(X_i, \tilde{A}_i) = (\tilde{C}_i^\star, \text{faithful}_i).$$

Only valid and faithful deployed headlines are evaluated. For valid headlines, the noise-free deployed reward is

$$v_i(\tilde{A}_i) = m^\star(Z_i^\star, \tilde{C}_i^\star).$$

In the main code-sufficient benchmark, this reduces to

$$v_i(\tilde{A}_i) = M_i(\tilde{C}_i^\star).$$

Thus, the benchmark has an exact code-level oracle while deployed policies return generated headlines.

F OFFLINEPROMPTS BENCHMARK

Beyond the two controlled benchmarks, we evaluate the complete method on the public MovieLens task from OfflinePrompts (Kiyohara et al., 2025), providing additional validation in an externally defined generative setting. Contexts combine the released embeddings of real MovieLens users and movies; actions are raw movie-related sentences generated by the frozen Mistral-7B-Instruct-v0.2 model, with no injected simulator features. Rewards come from the released personalized DistilBERT reward simulator trained on real user ratings. POLICYLIFT learns from context, generated text, and reward without observing the generating prompt.

Behavior system and logged data. For reproducible and computationally efficient evaluation, we pre-generate the Mistral outputs under the benchmark prompt set and reuse the same generated outputs across methods. We use a fixed subset of 500 movies and 100 prompts. For each movie–prompt pair, we generate one sentence by greedy decoding with a maximum of 30 new tokens and the released prompt formatter. The behavior system samples a prompt from a context-dependent softmax distribution and returns the corresponding pre-generated sentence. Its scores are bilinear in the context and prompt embeddings; we calibrate the softmax concentration on training contexts so that the top 50 prompts carry mean probability mass 0.90. Prompt identities and their logging probabilities are hidden from POLICYLIFT, which receives only context, sentence embedding, and reward. Each of five seeds produces 10,000 logged observations, adding independent Gaussian reward noise with standard deviation 0.05.

Representation and evaluation. We follow the released sentence-embedding procedure, projecting pooled Mistral hidden states to 20 dimensions with the supplied PCA matrix. We use the released user split, with disjoint training and test halves, and evaluate all methods on the same 10,000 sampled test contexts. The movie and prompt catalog is shared across the split. All learned policies and generated-sentence selections are frozen before evaluation with the reward simulator. Reported values average its expected rewards, without added observation noise. Behavior draws use the same behavior system for every method; POLICYLIFT uses a cap of $T_{\max} = 20$ and an independent behavior draw on fallback.

Training configuration. We use $K = 8$, $\tau = 0.03$, two hidden layers of width 64, batch size 32, learning rate 10^{-4}, and 50 Stage 1 epochs. The proposed configuration uses $\beta = 10^{-4}$ and $\gamma = 0.2$; predictive-only codes set both to zero. Stage 2 fits a ridge outcome model with context–code interactions and a multinomial logistic propensity model. RA selects the highest predicted reward among supported codes. Standard errors describe variability across these runs on the fixed test contexts and pre-generated outputs.

G ADDITIONAL RESULTS

G.1 SYNTHETIC DIAGNOSTICS

Figure 3 presents the complete synthetic diagnostics. Its columns vary within-code reward effects (α_{dep}, left), predictive signal (α_{pred}, middle), and logging-policy concentration (κ_{ov}, right). The left legend lists deployment rules; the right legend lists code representations for the middle and right columns. Predictiveness and overlap curves show means with standard errors over 5 seeds. Deployment values are computed analytically, so no standard errors are shown. The protocols are described in Appendix D.4.

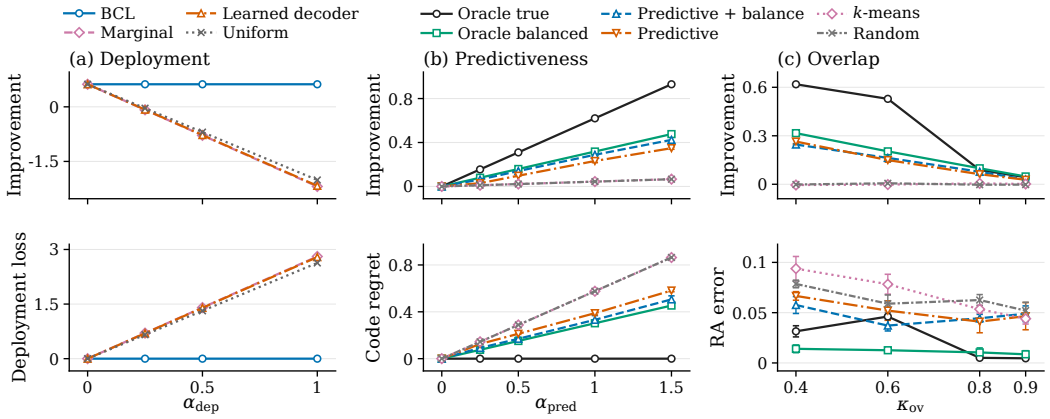


Figure 3: **Complete synthetic diagnostics.** Top: policy improvement (higher is better). Bottom: deployment loss, code regret, and RA estimation error, respectively (lower is better).

G.1.1 DEPLOYMENT

In the left column of Figure 3, ground-truth codes and the oracle code-level policy are fixed while the instantiation rule varies. Table 10 reports the strongest residual-effect setting, $\alpha_{\text{dep}} = 1$, under the default synthetic DGP. All rows use the ground-truth code $C^\star$, the oracle code-level policy, and the true support rule; rows differ only in the Stage 3 instantiation rule. Values are computed analytically under the known DGP, so no standard errors are reported.

Deployment loss is $V_C(\pi_C^\star) - V(\pi_\kappa)$, and absolute deployment error is $|V(\pi_\kappa) - V_C(\pi_C^\star)|$. Behavior-consistent lifting (BCL) preserves the code-level value, while alternative instantiation rules change the within-code action distribution and therefore incur deployment loss.

Table 10: **Deployment diagnostic.** Default synthetic DGP at $\alpha_{\text{dep}} = 1$; analytically computed values.

Deployment rule	Policy improvement $V(\pi) - V(\mu)$	Value $V(\pi)$	Deployment loss $V_C(\pi_C) - V(\pi_\kappa)$	Abs. deployment error
BCL / Ours	0.620	0.938	0.000	0.000
Marginal	-2.188	-1.870	2.808	2.808
Learned dec.	-2.170	-1.853	2.790	2.790
Uniform	-2.010	-1.693	2.630	2.630

G.1.2 PREDICTIVENESS

In the middle column of Figure 3, oracle Stage 2 optimization and behavior-consistent lifting are fixed while the representation varies. Table 11 reports the default synthetic DGP at $\alpha_{\text{pred}} = 1$. All rows use an oracle Stage 2 code-level policy and behavior-consistent lifting; rows differ only in the Stage 1 code representation. Values are reported as mean (standard error) over random seeds.

Oracle true codes give the learner-inaccessible upper benchmark, while oracle balanced codes are a designed support-aware proxy. Policy improvement is $V_G^\star - V(\mu)$, where $V_G^\star$ is the value of the oracle code-level policy for representation G. Code regret is $V_{C^\star}^\star - V_G^\star$. Prediction error is the noise-free

grouped error $\mathbb{E}[(Y - m_G(X,G))^2]$. Because Stage 2 is oracle for every row, differences isolate the information contained in the code representation.

Table 11: **Predictiveness diagnostic.** Default synthetic DGP at $\alpha_{\text{pred}} = 1$; mean (standard error) over random seeds.

Code representation	Value	Policy improvement $V(\pi) - V(\mu)$	Code regret $V^\star_{C^\star} - V^\star_G$	Prediction error $\mathbb{E}[(Y - m_G(X,G))^2]$
Oracle true codes	1.000 (0.000)	0.620 (0.000)	0.000 (0.000)	0.000 (0.000)
Oracle balanced codes	0.698 (0.000)	0.318 (0.000)	0.302 (0.000)	0.012 (0.000)
Pred.+bal. codes	0.668 (0.036)	0.288 (0.036)	0.332 (0.036)	0.088 (0.001)
Pred. codes	0.612 (0.015)	0.232 (0.015)	0.388 (0.015)	0.084 (0.003)
K-means	0.424 (0.006)	0.044 (0.006)	0.576 (0.006)	0.148 (0.001)
Random	0.424 (0.004)	0.044 (0.004)	0.576 (0.004)	0.148 (0.001)

G.1.3 Overlap

In the right column of Figure 3, behavior-consistent lifting and the support constraint are fixed while representations are compared under increasing logging-policy concentration. Together, the overlap panels distinguish attainable policy improvement from value-estimation accuracy.

Table 12 reports the default synthetic DGP at $\kappa_{\text{ov}} = 0.9$. Rows use the same code representations as in Table 11, while Stage 2 is learned with a regression-adjusted objective and all rows use the τ support constraint and behavior-consistent lifting. Values are reported as mean (standard error) over random seeds.

Oracle true codes give the learner-inaccessible upper benchmark under the support constraint, while oracle balanced codes are a designed support-aware proxy. RA error is $\left|\widehat{V}_{\text{RA}}(\widehat{\pi}_C) - V_C(\widehat{\pi}_C)\right|$. Expected draws is $\mathbb{E}[1/\mu_C(\widehat{c}(X) \mid X)]$ and measures the cost of behavior-consistent lifting. All rows are support constrained, so unsupported-code rates are zero by construction and omitted.

Table 12: **Overlap diagnostic.** Default synthetic DGP at $\kappa_{\text{ov}} = 0.9$; mean (standard error) over random seeds.

Code representation	Value	Policy improvement $V(\pi) - V(\mu)$	RA error $\lvert\widehat{V}_{\text{RA}} - V\rvert$	Expected draws
Oracle true codes	0.649 (0.000)	0.044 (0.000)	0.005 (0.000)	1.10
Oracle balanced codes	0.653 (0.000)	0.048 (0.000)	0.009 (0.004)	2.16
Pred.+bal. codes	0.635 (0.008)	0.030 (0.008)	0.049 (0.008)	3.63
Pred. codes	0.633 (0.004)	0.028 (0.004)	0.047 (0.014)	3.09
K-means	0.607 (0.003)	0.002 (0.003)	0.044 (0.005)	7.96
Random	0.603 (0.002)	-0.002 (0.002)	0.052 (0.008)	8.17

G.2 Sensitivity to loss parameters

Figure 4 shows sensitivity to the loss parameters β (top) and γ (bottom) on the semi-synthetic dataset. The left panels report policy improvement (higher is better), and the right panels report expected draws (lower is better).

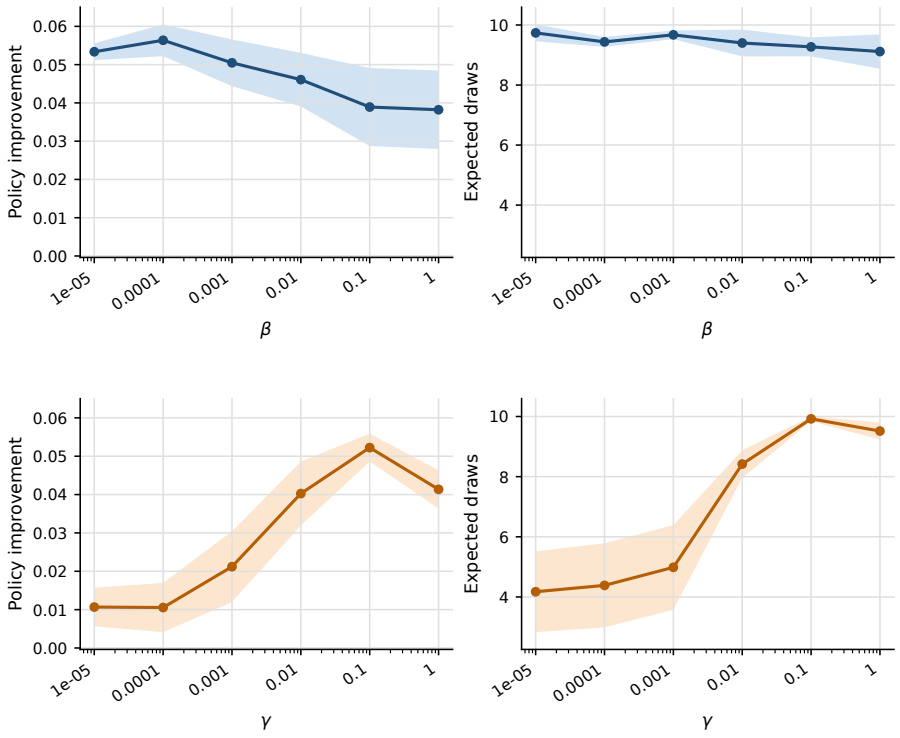


Figure 4: **Sensitivity to loss parameters on the semi-synthetic dataset.** β varies in the top row and γ in the bottom row. Left: policy improvement; right: expected draws.

H Implementation details

Synthetic diagnostics. We follow the data generation and evaluation as described in D. Table 13 gives the implementation choices for the reported runs. We provide code of our method in `https://anonymous.4open.science/r/GenerativeOPL-465A/README.md`

Semi-synthetic headline benchmark. The semi-synthetic experiments use the logged headlines, validation procedure, and candidate pool from Appendix E. Table 14 gives the implementation choices for the reported runs.

H.1 Implementation of the experimental comparisons

Code substitutions. All configurations use $K = 8$ codes. Random codes provide an outcome-independent comparison: synthetic and OfflinePrompts actions receive a fixed random-projection assignment with quantile bins fitted on logged training actions; headlines use seeded balanced random assignments. The k-means alternative clusters action features without reward supervision. Predictive-only codes retain the learned architecture and prediction loss but set $\beta = \gamma = 0$. Code substitutions retain the RA objective and the same evaluation protocol. For objective substitutions, all methods share the proposed encoder within each seed.

Stage 2 objectives. Our Code-IPS objective applies inverse propensity weighting to the observed code, following the marginalization principle of MIPS (Saito & Joachims, 2022). Code-DR adds a code-level reward prediction and an importance-weighted residual correction, motivated by regression-assisted estimation such as OffCEM (Saito et al., 2023). Kernel-IS adapts the sentence-level smoothing idea of Kiyohara et al. (2025): Gaussian distances from action features to code centroids define normalized soft memberships, and a soft-label propensity model estimates their conditional means. We use the median-distance bandwidth heuristic. The kernel objective weights rewards by policy

Algorithm 2: PolicyLift (implementation-aligned)

Input: Logged data $\mathcal{D} = \{(X_i, A_i, Y_i)\}_{i=1}^n$; code encoder g_ϕ (action branch), context encoder ψ, outcome head f_θ; number of codes K; weights $\beta \geq 0, \gamma \geq 0$; threshold $\tau \geq 0$; acceptance cap $T_{\max}$.

Output: Deployed policy sampler $\widehat{\pi}^{\uparrow}(\cdot \mid x)$.

Stage 1: Learn code encoder and predictor;
Initialize (ϕ, ψ, θ);
while *not converged* **do**
  Compute logits $\ell_i \leftarrow \mathrm{enc}_a(A_i)$ and straight-through Gumbel-hard codes $\tilde{C}_i \leftarrow \mathrm{GumbelSoftmax}(\ell_i; \texttt{hard=true})$;
  Compute context embedding $Z_i \leftarrow \psi(X_i)$;
  $\widehat{\mathcal{L}}_{\text{pred}} \leftarrow \frac{1}{n}\sum_i \left(Y_i - f_\theta([Z_i, \tilde{C}_i])\right)^2$;
  $\widehat{\mathcal{L}}_{\text{ov}} \leftarrow \sum_{c=1}^K \widehat{D}\Big(\widehat{P}_Z, \widehat{P}_{Z|\tilde{C}=c}\Big)$; `// stop-gradient through Z in this term`
  $\widehat{\mathcal{L}}_{\text{cb}} \leftarrow \mathrm{KL}(\widehat{p}_C \| \mathrm{Unif}(\{1, \ldots, K\}))$, with $\widehat{p}_C = \frac{1}{n}\sum_i \tilde{C}_i$;
  Update (ϕ, ψ, θ) to minimize $\widehat{\mathcal{L}}_{\text{pred}} + \beta\, \widehat{\mathcal{L}}_{\text{ov}} + \gamma\, \widehat{\mathcal{L}}_{\text{cb}}$;
Freeze Stage-1 model; define hard codes for Stage 2 by $C_i = \arg\max g_\phi(A_i)$;
Stage 2: Learn outcome head, propensity proxy, and code policy;
Fit $\widehat{m}(x, c) \approx \mathbb{E}[Y \mid X = x, C = c]$ on (X_i, C_i, Y_i) (neural head; optional initialization from Stage-1 head);
Fit $\widehat{\mu}_C(c \mid x)$ via multiclass cross-entropy on (X_i, C_i);
Define $\pi_C(x) \in \arg\max_{c:\, \widehat{\mu}_C(c|x) \geq \tau} \widehat{m}(x, c)$;
Stage 3: Deployment at context x;
Select target code $c^\star = \pi_C(x)$;
for $t = 1, \ldots, T_{\max}$ **do**
  Sample $a_t \sim \mu(\cdot \mid x)$;
  if $\arg\max g_\phi(a_t) = c^\star$ **then**
    return a_t
return *sample from behavior* $\mu(\cdot \mid x)$; `// fallback after cap`

Table 13: Implementation details for the synthetic diagnostics.

Component	Choice
Code space	$K = 8$
Context/action dimensions	12 and $8 + 8 + 4 + 64 + 2$
Support threshold	$\tau = 0.03$
Deployment diagnostic	oracle codes and oracle code policy; vary only deployment rule
Predictiveness diagnostic	BCL and oracle Stage 2; vary code representation and α_{pred}
Overlap diagnostic	learned RA Stage 2 with support rule; vary representation and κ_{ov}
POLICYLIFT Stage 1	MLP hidden sizes $(64, 64)$, latent dimension 8, Gumbel temperature 0.5
Optimization	50 epochs, batch size 32, learning rate 10^{-4}
Main regularization	$\beta = 0.0001$, $\gamma = 0.2$ for Pred.+bal.
Reported uncertainty	mean with one standard error across seeds

probabilities times these memberships divided by their estimated conditional means. For all three objectives, inverse propensities are capped at $1/\tau$, where $\tau = 0.03$.

We optimize linear softmax code policies on standardized context features and deploy their highest-scoring supported code. RA instead maximizes the fitted code-level outcome directly. IPS and DR use hard-code propensities; kernel-IS uses soft-membership propensities for its objective and support on the text benchmarks. The synthetic comparison uses the common hard-code support at deployment for every objective. Synthetic support is computed from grouped simulator frequencies

Table 14: Implementation details for the semi-synthetic headline benchmark.

Component	Choice
Data	905 logged actions, 2513 generated actions, 5 seeds
Data generator model	GPT-5.4-mini
Code space and support	$K = 8$, $\tau = 0.03$
Reward	strategy-rank lookup with $(1.00, 0.65, 0.15, -0.15)$ and context baseline scale 0.2
Context representation	sentence-transformers/all-MiniLM-L6-v2; append $3.0\, e_{Z^\star}$
Action representation	sentence-transformers/all-MiniLM-L6-v2; append noisy $C^\star$, strategy, nuisance, and random blocks
Action-block parameters	$(0.8, 0.15)$ for $C^\star$, $(0.6, 0.25)$ for strategy, nuisance dim 64 with scale 4.0 and noise 0.5
POLICYLIFT Stage 1	MLP hidden sizes $(64, 64)$, latent dimension 8, Gumbel temperature 0.5
Optimization	50 epochs, batch size 32, learning rate 10^{-4}
Main regularization	$\beta = 0.0001$, $\gamma = 0.2$ for Pred.+bal.

conditional on the latent context group, following the controlled diagnostic protocol; headline and OfflinePrompts propensities are fitted from logged data. Within each dataset, the baselines share the candidate generator and deployment evaluator.

Main table: configurations and evaluation. The synthetic column uses $\alpha_{\text{pred}} = 1$, $\alpha_{\text{dep}} = 0$, $\kappa_{\text{ov}} = 0.6$, 5,000 logged observations, and 10,000 noise-free evaluation samples. Its Stage 1 network has two hidden layers of width 32, learning rate 10^{-3}, 50 epochs, batch size 32, and $(\beta, \gamma) = (0, 0.001)$. We evaluate ideal BCL through grouped simulator conditional rewards. The headline column retains MiniLM features alongside the constructed context and action feature blocks described in Appendix E; it uses width 64, learning rate 10^{-4}, 50 epochs, batch size 32, and $(\beta, \gamma) = (10^{-4}, 0.2)$. We perform runs across five random seeds.

Prompt-based reference methods. On OfflinePrompts, *prompt-based regression* fits a neural reward predictor to context and logged prompt embeddings, then selects the highest-scoring prompt from the 100-prompt catalog. The *DSO-style prompt policy* learns a context-dependent distribution over that 100-prompt catalog using kernel importance weighting in sentence-embedding space. It uses the logging distribution and an exact sum over the pre-generated prompt outputs to compute the kernel-smoothed logging marginal; importance ratios are clipped at 200. The bandwidth is the median within-movie sentence-embedding distance. Both baselines use the same generated sentences, logged reward data, test contexts, and reward simulator as POLICYLIFT, but additionally have access to prompt information that is unavailable to POLICYLIFT. In particular, prompt-based regression observes logged prompt embeddings, while the DSO-style policy accesses the prompt catalog, its generated outputs, and the logging distribution.

Compute resources. Experimental runs were conducted on commodity CPU hardware without GPU acceleration, with typical runs completing in under one hour. The semi-synthetic benchmark additionally required OpenAI API calls for headline generation and validation during dataset construction, which also completed within approximately one hour.