AI4AI IN UNLEARNING: IMPROVING UNLEARNING ALGORITHM AUTODESIGN WITH UDSL

Anonymous authors
Paper under double-blind review

ABSTRACT

Machine unlearning requires removing the influence of selected data while preserving utility on retained data. Existing unlearning methods are typically implemented as monolithic trainers, where choices about what signal to optimize, how to update the model, where to apply the update, and what behavior to protect are intertwined in code. This representation makes it difficult to systematically explore alternative combinations of unlearning mechanisms. We introduce **UDSL**, a domain-specific language that factorizes an unlearning procedure into four composable dimensions—**Signal**, **Update**, **Scope**, and **Protect**—and compiles each program onto a shared training stack. This representation enables LLM-guided evolutionary search to explore algorithmic structure rather than implementation details. On TOFU-forget10 with Llama-3.2-1B-Instruct, UDSL-guided search discovers unlearning procedures that outperform several established unlearning methods under a small evaluation budget. Ablations show that the gains stem from exploring new compositions of unlearning mechanisms rather than merely tuning or recombining existing methods. Across forget ratios and datasets, UDSL provides a common representation in which new procedures can be re-searched without redesigning the underlying training infrastructure. These results suggest that, for LLM-guided unlearning design, the representation of the algorithmic search space can be as important as the search procedure itself.

1 INTRODUCTION

Machine unlearning aims to remove the influence of selected training data from a trained model while preserving its behavior on the remaining data (Cao & Yang, 2015; Guo et al., 2020). As the field has progressed, a growing number of unlearning algorithms have been proposed, including optimization-based, preference-based, and model-specific approaches (Nguyen et al., 2025; Zhang et al., 2024; Fan et al., 2025; Eldan & Russinovich, 2023; Yao et al., 2024). Although these methods share the same high-level objective, they differ substantially in how forgetting is formulated and executed. In particular, an unlearning procedure must determine *what signal identifies information to forget*, *how that signal induces parameter updates*, *where those updates are applied*, and *how retained knowledge is protected*. These interacting choices define a rich algorithmic design space, in which different combinations can produce substantially different utility–forgetting trade-offs.

Existing methods, however, expose this design space primarily through fixed, human-designed recipes. Each published trainer encodes one particular combination of algorithmic decisions, while alternative combinations remain implicit in implementation code. Even when different methods share reusable mechanisms, these components are rarely represented as explicit and composable search variables. The space of potentially useful unlearning procedures is therefore substantially larger than the catalogue of named algorithms.

Automated algorithm design provides a natural way to explore this space. Recent AI-for-AI (AI4AI) systems have shown that large language models can generate and evolve executable programs, using empirical evaluation to guide the search toward increasingly effective algorithms (Liu et al., 2024; Ye et al., 2024; Novikov et al., 2025; Yang et al., 2026). For machine unlearning, such a paradigm could replace manual enumeration of mechanism combinations with an automated loop of proposing, evaluating, and refining unlearning procedures.

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.

Yet applying automated program search to unlearning raises a fundamental question: *what should the search object be?*

A straightforward choice is to let an LLM directly modify executable trainer code. While flexible, this representation entangles algorithmic decisions with implementation details. A mutation may alter a meaningful forgetting mechanism, but it may also modify training-loop structure, data handling, or other low-level code without producing a meaningful algorithmic change. More fundamentally, the representation determines which algorithmic structures can be expressed and efficiently explored. A search procedure cannot reliably discover a useful composition that its representation does not make writable.

This motivates moving the search interface from implementation space to the *algorithmic design space of machine unlearning*. We introduce **UDSL**, a domain-specific language that represents an unlearning procedure through four semantic dimensions: **Signal**, **Update**, **Scope**, and **Protect**. Signal specifies the information used to drive forgetting; Update determines how this signal is translated into parameter changes; Scope specifies where and at what granularity the modification is applied; and Protect specifies how retained behavior is preserved. Rather than treating an unlearning trainer as an indivisible implementation, UDSL makes these algorithmic choices explicit and composable.

UDSL supports recursive composition of these primitives, allowing existing human-designed trainers to be represented within the same structured space while also expressing procedures that do not correspond directly to any single named method. An interpreter maps each UDSL program to an executable unlearning procedure using the same underlying training stack as the baselines. This separation allows automated search to modify algorithmic structure while keeping implementation and evaluation consistent. The central role of UDSL is therefore not merely to provide a compact program representation, but to expose meaningful algorithmic structure as a writable search space.

We use UDSL as the search space for LLM-guided evolutionary discovery (Figure 1). The LLM proposes semantic mutations to previously evaluated UDSL programs, while empirical fitness determines which candidates are retained for subsequent exploration. In this formulation, evolution searches over explicit combinations of unlearning mechanisms rather than arbitrary trainer implementations.

We evaluate UDSL-guided search across unlearning settings involving different datasets, forgetting ratios, and model configurations. On TOFU-forget10 with Llama-3.2-1B-Instruct, UDSL-guided search discovers procedures that outperform several established unlearning methods under a small evaluation budget. Further experiments examine whether discovered procedures transfer across settings and whether the representation itself can support efficient re-search when the setting changes. Ablation studies isolate the contribution of compositional structure from coefficient tuning and recombination of existing named methods. Together, these experiments investigate whether exposing algorithmic structure through UDSL enables automated discovery beyond the space of existing hand-designed recipes.

Our contributions are summarized as follows:

- We introduce **UDSL**, a compositional domain-specific language for machine-unlearning algorithm design that factorizes unlearning procedures into Signal, Update, Scope, and Protect, and supports recursive composition of these mechanisms.
- We establish an **LLM-guided evolutionary search interface** over this structured space, separating algorithmic search from implementation through an interpreter and enabling automated discovery of unlearning procedures.
- We provide an **empirical study of representation-driven unlearning autodesign**, comparing UDSL against established methods and direct LLM-based approaches, and analyzing compositionality, transfer, and re-search across datasets and forgetting settings.

Overall, our work treats machine unlearning algorithm design as a structured search problem: the effectiveness of automated discovery depends not only on how the search is performed, but also on what the representation allows the search to express.

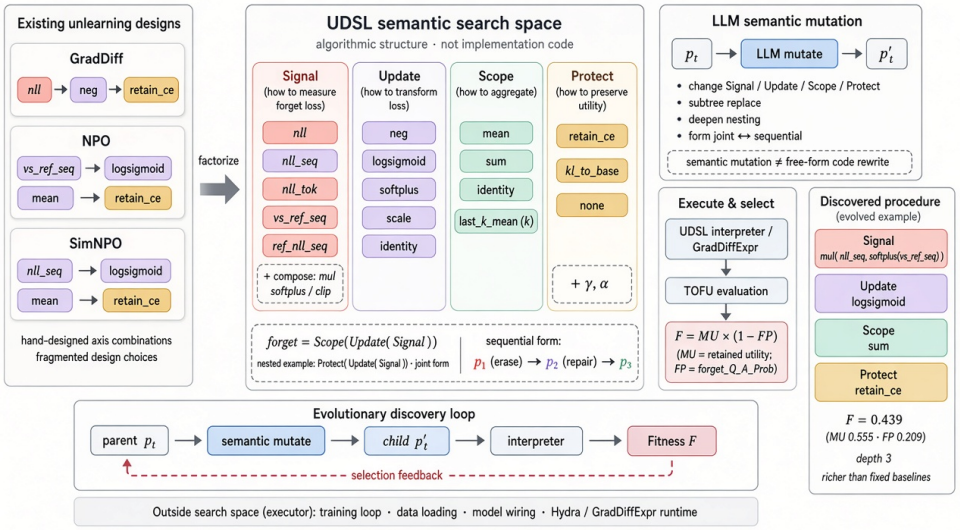

Figure 1: UDSL search interface. Existing trainers are factorized into Signal, Update, Scope, and Protect. An LLM mutates a structured program; an interpreter compiles it onto the same stack as published baselines; fitness $\mathrm{MU} \times (1 - \mathrm{FP})$ selects parents. The highlighted program is the TOFU-forget10 discovery (main-text Fitness 0.442; Appendix E).

2 Related Work

Human designed machine unlearning algorithms. Machine unlearning aims to remove the influence of selected training data while preserving utility on retained data (Cao & Yang, 2015; Guo et al., 2020). Existing methods instantiate this goal through diverse algorithmic mechanisms, including optimization- and preference-based objectives (Zhang et al., 2024; Fan et al., 2025), approximate or LLM-oriented unlearning procedures (Eldan & Russinovich, 2023; Yao et al., 2024), and broader families surveyed in prior work (Nguyen et al., 2025). Across these designs, unlearning procedures make different choices about how information is identified for forgetting, how parameters are updated, where modifications are applied, and how retained knowledge is protected. Although such methods explore a rich set of mechanisms, these algorithmic components are largely specified and evaluated as manually designed procedures. As a result, the underlying design space contains many meaningful choices, yet they are typically exposed through fixed recipes rather than a common compositional representation for systematic search.

LLM-guided evolutionary program search. Recent work increasingly treats large language models as semantic operators within evolutionary program search. EoH uses LLMs to generate and evolve heuristic programs (Liu et al., 2024), while ReEvo incorporates LLM-generated reflection as feedback for subsequent revisions (Ye et al., 2024). AlphaEvolve extends this paradigm to broader algorithmic and scientific discovery by evolving executable programs against automated evaluators (Novikov et al., 2025), and subsequent systems such as TurboEvolve further emphasize sample efficiency and robust search dynamics (Yang et al., 2026). Collectively, these approaches share a common abstraction: the LLM proposes or modifies candidate programs, while executable evaluation provides the signal for evolutionary selection. Despite substantial progress in mutation, reflection, population management, and evaluation efficiency, the underlying program representation is often inherited from the application setting and treated as fixed. Existing systems therefore primarily advance the search process, while leaving comparatively less explicit how the algorithmic search space itself should be represented.

Structured search spaces and DSL-based program synthesis. A complementary line of work studies structured search spaces and domain-specific program representations. DSLs and library-learning methods encode domain-relevant abstractions that reduce the effective synthesis space (Bowers et al., 2023), and semantic-space search can avoid redundant exploration of syntactically distinct but behaviorally similar programs (Moraes & Lelis, 2024). These studies suggest that structured representations can make meaningful algorithmic changes more explicit and provide more controllable directions for search. For machine unlearning, this motivates representing procedures through composable semantic dimensions such as Signal, Update, Scope, and Protect, so that evolutionary search can operate over meaningful algorithmic choices rather than only implementation-level variations. Our work brings these perspectives together through UDSL, which combines machine-unlearning algorithm design, LLM-guided evolutionary search, and a structured semantic representation for exploring unlearning procedures.

3 METHOD

3.1 PROBLEM FORMULATION

We formulate machine unlearning following the certified removal framework of Guo et al. (2020). Let D be a training dataset and $\mathcal{A} : D \mapsto h$ be a learning algorithm that maps a dataset to a model $h \in \mathcal{H}$. Given a forgetting request for $x \in D$, the ideal objective is to obtain a model whose behavior is indistinguishable from that of a model trained from scratch on the remaining dataset D_{-x}.

More formally, a removal mechanism $\mathcal{M}$ provides ϵ-certified removal if, for all $T \subseteq \mathcal{H}$, datasets D, and samples $x \in D$,

$$\Pr\left[\mathcal{M}(\mathcal{A}(D), D, x) \in T\right] \leq e^{\epsilon} \Pr\left[\mathcal{A}(D_{-x}) \in T\right]. \tag{1}$$

We do not claim ϵ-certified removal. The remainder of the paper studies *approximate* unlearning: reduce the influence of forgotten data while preserving utility on retained data. Given a trained model h, a forget set D_f, and a retain set D_r, we express this trade-off abstractly as

$$\mathcal{L}_{\text{unlearn}} = \mathcal{L}_{\text{forget}}(h'; D_f) + \gamma\, \mathcal{L}_{\text{retain}}(h'; D_r), \tag{2}$$

where γ balances forgetting against retention. Instead of manually fixing the functional forms of these objectives, we search over executable unlearning procedures.

A candidate is a structured program p drawn from the UDSL search space $p \in \mathcal{P}_{\text{UDSL}}$. An interpreter $\mathcal{I}$ maps p to an executable procedure,

$$h' = \mathcal{I}(p, h, D_f, D_r), \tag{3}$$

and the resulting model is scored by the empirical fitness

$$F(p) = \text{MU}(p) \times \big(1 - \text{FP}(p)\big), \tag{4}$$

where MU measures utility on retained data and FP measures residual performance on forgotten data. Higher F indicates a better utility–forgetting trade-off. The discovery objective is to identify a high-fitness program among those explored,

$$p^* = \arg \max_{p \in \mathcal{P}_{\text{explored}}} F(p). \tag{5}$$

3.2 UDSL: SEMANTIC REPRESENTATION

UDSL is a domain-specific language for machine-unlearning algorithm design. It separates algorithmic structure from implementation details so that search can operate on interpretable design decisions rather than entangled training code. A UDSL program factorizes an unlearning procedure along four composable semantic dimensions: **Signal**,**Update**,**Scope**,**Protect**. Signal specifies the information or quantity used to drive forgetting; Update specifies how that signal is converted into parameter changes; Scope specifies where and at what granularity the modification is applied; and Protect specifies how retained behavior is preserved during forgetting. These dimensions expose semantically distinct algorithmic decisions as explicit search variables.

UDSL further supports composition and recursive nesting of primitives, so that mechanisms can be combined into deeper procedures rather than only selected from a flat catalogue of existing trainers. Representative primitives include preference- or contrastive-style forget signals, parameter updates, global or token-level scopes, KL-based protection, and sequential composition of potentially conflicting stages. A nested form such as Protect(Update(Signal(argument))) illustrates that UDSL defines a structured and compositional algorithmic search space. Implementation details of training loops, tokenization, and data handling are left to the interpreter and therefore remain outside the semantic search space. Abstractly, a program is a nested term

$$p \;::=\; \mathrm{Protect}\big(\mathrm{Update}(\mathrm{Signal}),\, \mathrm{Scope}\big) \;\mid\; \mathrm{Seq}(p, p) \;\mid\; \mathrm{named},$$

where named recovers a published trainer as a depth-1 point, and Seq allows staged or nested composition. The interpreter $\mathcal{I}$ in Eq. equation 3 maps every p onto the same optimizer, batching, and evaluation stack as the published baselines; learning rate, epoch count, and batch size are fixed in Section 4.1 and are not search variables unless an ablation explicitly unfreezes them.

3.3 LLM-Guided Evolution

We discover UDSL programs through LLM-guided evolutionary search. The LLM acts as a semantic mutation operator over $\mathcal{P}_{\mathrm{UDSL}}$, and empirical fitness grounds subsequent selection. Given a parent p_t selected from previously evaluated candidates according to observed fitness, the LLM proposes a mutated child

$$p'_t = \mathrm{Mutate}_{\mathrm{LLM}}(p_t) \tag{6}$$

that remains valid under the UDSL grammar. Invalid proposals are rejected and do not consume a successful-training evaluation. Mutations may revise Signal, Update, Scope, or Protect components, as well as composition structure or nesting depth. For example, $\mathrm{Update}(\mathrm{Signal}_1)$ may become $\mathrm{Update}(\mathrm{Protect}(\mathrm{Signal}_2))$, changing an algorithmic relationship rather than rewriting implementation details.

The child is interpreted by Eq. equation 3 and scored by Eq. equation 4. Fitness provides an empirical grounding signal: candidates are retained or discarded based on measured utility–forgetting trade-offs. Competitive programs are kept as potential parents for later mutations via

$$p_{t+1} = \mathrm{Select}\big(p_t,\, p'_t\big), \tag{7}$$

and after the evaluation budget is exhausted the best discovered procedure is returned as p^* in Eq. equation 5.

Under conventional code-level search, an LLM typically rewrites executable implementation text and the resulting program is then run directly. In our framework, the LLM instead mutates a structured UDSL program; the interpreter realizes that program as an executable unlearning procedure, and empirical fitness determines which candidates are retained. Search therefore operates on algorithmic structure, while implementation is handled by the interpreter. We evolve structured UDSL programs whose semantic structure corresponds to algorithmic decisions in machine unlearning, with implementation details kept outside the search space.

4 Experiments

4.1 Experimental Setup

We evaluate UDSL on TOFU (Maini et al., 2024) with Llama-3.2-1B-Instruct unless otherwise stated, using forget10 / retain90, five unlearning epochs, AdamW at $1{\times}10^{-5}$, and batch size $2{\times}4$. Every published trainer and every UDSL program is compiled onto the same open-unlearning stack to ensure a consistent implementation and evaluation protocol.

On TOFU, we report model utility (MU; ↑) on retained data, forget probability (FP; ↓) as `forget_Q_A_Prob`, forget ROUGE (FR; ↓) as `forget_Q_A_ROUGE`, and extraction strength (ES; ↓). Our primary search metric is Fitness $= \mathrm{MU} \times (1 - \mathrm{FP})$.

Fitness is used for search and selection, while FR and ES provide additional leakage readouts from the same evaluation logs and are not used for selection. We do not tabulate official Forget Quality

(KS), which is near zero for every method in these experiments, or PrivLeak, which is too noisy to provide a stable ranking of trainers. We report Δ_{GD} as the difference between a method's Fitness and GradDiff on the same split.

For MUSE (Shi et al., 2025), Forget is the mean of KnowMem and VerbMem on forgotten text, while Utility is KnowMem on retained text. For WMDP-cyber (Li et al., 2024), Forget is hazardous-split accuracy and Utility is MMLU. We use $\mathrm{Score} = \mathrm{Utility} \times (1 - \mathrm{Forget})$ for these benchmarks.

We distinguish two LLM proposal settings. In the *HD* setting, the LLM chooses a named unlearning method and its scalar hyperparameters. In the *UDSL* setting, the LLM writes a structured UDSL program. A *13-evaluation search* denotes a search with a fixed budget of 13 completed training-and-evaluation runs. Unless otherwise stated, comparisons between search procedures use this same evaluation budget.

Our experiments address four questions: (1) whether UDSL search can discover procedures that outperform established unlearning methods; (2) whether a discovered procedure can adapt across forgetting ratios and model scales; (3) whether the same representation remains useful across datasets; and (4) which aspects of the UDSL representation are responsible for its search advantage.

4.2 Main Results: Can UDSL Discover Better Unlearning Procedures?

We first ask whether searching directly in UDSL can discover procedures that improve upon existing human-designed trainers, and whether UDSL is a more effective proposal language than asking an LLM to select and configure a named trainer. Table 1 compares published trainers (Dong et al., 2025; Yang et al., 2025; Yuan et al., 2025), one-shot LLM proposals, and a 13-evaluation UDSL search on TOFU-forget10.

Table 1: TOFU-forget10 with Llama-3.2-1B-Instruct. Fitness $= \mathrm{MU} \times (1 - \mathrm{FP})$. FR = forget ROUGE; ES = extraction strength. HD = named-trainer proposal; UDSL = structured program. Each one-shot row uses a single proposal, while UDSL search uses a fixed 13-evaluation budget.

Method	**MU↑**	**FP↓**	**FR↓**	**ES↓**	**Fit.↑**	**Method**	**MU↑**	**FP↓**	**FR↓**	**ES↓**	**Fit.↑**
Published						*LLM*					
NPO	0.488	0.212	**0.245**	0.099	0.385	GPT-5.5 HD	0.482	0.222	0.263	**0.078**	0.375
SimNPO	**0.596**	0.843	0.725	0.575	0.093	GPT-5.5 UDSL	0.495	0.153	0.387	0.103	0.420
GradDiff	0.439	0.066	0.347	0.086	0.409	GPT-5.6 HD	0.447	**0.060**	0.363	0.079	0.420
UNDIAL	0.576	0.423	0.473	0.177	0.332	GPT-5.6 UDSL	0.443	0.063	0.343	0.084	0.415
SatImp	**0.596**	0.878	0.785	0.676	0.073	Claude-Opus-4.6 HD	0.594	0.482	0.502	0.222	0.308
WGA	**0.596**	0.541	0.473	0.177	0.273	Claude-Opus-4.6 UDSL	0.568	0.263	0.425	0.148	0.419
ME+GD	0.574	0.663	0.521	0.298	0.194	Claude-Opus-4.7 HD	0.485	0.202	0.256	0.086	0.387
IDK+AP	0.544	0.605	0.373	0.272	0.215	Claude-Opus-4.7 UDSL	0.515	0.150	–	–	0.438
UDSL search (13 evaluations): $\mathrm{MU} = 0.554$, $\mathrm{FP} = 0.201$, $\mathrm{FR} = 0.414$, $\mathrm{ES} = 0.138$, **Fitness=**0.442											

The published trainers illustrate the utility–forgetting trade-off. Several methods retain high utility but exhibit weak forgetting, while GradDiff provides the strongest Fitness among the compared published trainers. This motivates evaluating the complete trade-off rather than optimizing utility or forgetting in isolation.

The one-shot comparison shows that UDSL is generally a stronger proposal interface than selecting a named trainer with scalar hyperparameters. UDSL proposals outperform their HD counterparts for GPT-5.5 and both Claude models, while GPT-5.6 is approximately tied. In particular, Claude-Opus-4.6 + UDSL reaches Fitness 0.419 compared with 0.308 for the corresponding HD proposal, while Claude-Opus-4.7 + UDSL reaches 0.438. These results suggest that exposing algorithmic structure gives the LLM a more expressive proposal space than choosing among named trainers alone.

With the same UDSL representation, a 13-evaluation search reaches Fitness 0.442, exceeding GradDiff (0.409). The discovered procedure uses a compositional sequence involving a sequence-level NLL signal gated by $\mathrm{softplus}(\mathrm{vs_ref})$, a logistic update, token aggregation, and retain CE; the complete program is given in Appendix E. The resulting program contains nested composition, but the exact program depth is not treated as a search dimension or evaluation axis in this work.

Independent redraws of the same one-shot prompt exhibit substantial variance: Claude-Opus-4.7 + UDSL reaches 0.438 in the reported run, while later redraws can collapse. We therefore report representative one-shot proposals rather than averaging over multiple stochastic redraws; additional repeats are provided in Appendix D.

Takeaway. UDSL provides a proposal language in which an LLM can express unlearning procedures beyond named-trainer selection, and a small-budget search in this space can discover a procedure that exceeds the strongest compared published trainer on TOFU-forget10.

4.3 Adaptation Across Forgetting Ratios and Model Scales

A discovered procedure need not be universally transferable for the representation to remain useful. We therefore ask whether the TOFU-forget10 program can be directly migrated to different forgetting ratios and model scales, and whether re-searching within the same UDSL representation can adapt to the new setting. Table 2 compares the zero-shot migration of the forget10 program with an independent 13-evaluation search and GradDiff.

Table 2: Each setting uses the same four conditions: SimNPO, GradDiff, zero-shot migration of the forget10 UDSL program, and a 13-evaluation UDSL search. Migrate uses the forget10 winner without additional search. At 1B/10%, migration and search correspond to the original discovery. The 3B SimNPO row uses the UDSL SimNPO parent. Δ_{GD} is Fitness minus GradDiff on the same split.

Setting	Method	Mode	MU↑	FP↓	Fit.↑	Δ_{GD}
1B / 1%	SimNPO	published	**0.593**	0.858	0.084	−0.292
1B / 1%	GradDiff	published	0.566	**0.336**	**0.376**	0
1B / 1%	UDSL	migrate	0.556	0.355	0.359	−0.017
1B / 1%	UDSL	search	0.572	0.363	0.364	−0.012
1B / 5%	SimNPO	published	**0.595**	0.853	0.088	−0.329
1B / 5%	GradDiff	published	0.474	**0.120**	**0.417**	0
1B / 5%	UDSL	migrate	0.545	0.268	0.399	−0.018
1B / 5%	UDSL	search	0.515	0.306	0.357	−0.060
1B / 10%	SimNPO	published	**0.596**	0.843	0.093	−0.316
1B / 10%	GradDiff	published	0.439	**0.066**	0.409	0
1B / 10%	UDSL	migrate	0.554	0.201	**0.442**	+0.033
1B / 10%	UDSL	search	0.554	0.201	**0.442**	+0.033
3B / 10%	SimNPO	published	0.626	0.231	0.482	−0.013
3B / 10%	GradDiff	published	0.539	0.082	0.495	0
3B / 10%	UDSL	migrate	**0.666**	0.951	0.033	−0.462
3B / 10%	UDSL	search	0.655	**0.048**	**0.623**	+0.128

The forget10 program does not transfer reliably across settings. At 1% and 5%, migration improves over SimNPO but remains slightly below GradDiff, while a 13-evaluation re-search does not close the gap. At 3B/10%, direct migration collapses the utility–forgetting trade-off, whereas re-search recovers a substantially stronger procedure and exceeds GradDiff.

These results distinguish two forms of reuse. The discovered *program* is setting-specific, but the UDSL *representation* can be reused to search for a new procedure. Importantly, re-search is not guaranteed to succeed under a fixed budget: the 1% and 5% searches remain below GradDiff. Thus, the result is not that UDSL produces a universally transferable unlearning algorithm, but that it provides a common search interface across changing unlearning settings.

4.4 Adaptation Across Datasets

We next ask whether the same representation remains useful when the unlearning task changes. Table 3 compares GradDiff, zero-shot migration of the TOFU program, and an independent 13-evaluation UDSL search across MUSE-News, MUSE-Books, WMDP-cyber, ConceptGuard (Kale & Harris, 2026), and RWKU (Cao et al., 2024).

Table 3: Cross-dataset Score $=$ Utility $\times$ $(1 -$ Forget$)$. Mode distinguishes a published trainer, TOFU-program migration, and per-dataset UDSL search. Δ_{GD} is Score minus GradDiff on the same dataset.

Dataset	Method	Mode	Forg.↓	Util.↑	Score↑	Δ_{GD}
TOFU-forget10	GradDiff	published	**0.066**	0.439	0.409	0
	UDSL	search	0.201	0.554	**0.442**	+0.033
MUSE-News	GradDiff	published	0.556	0.520	0.231	0
	UDSL	migrate	0.579	0.529	0.223	−0.008
	UDSL	search	**0.462**	**0.539**	**0.290**	+0.059
MUSE-Books	GradDiff	published	**0.000**	0.110	0.110	0
	UDSL	migrate	0.220	0.607	0.473	+0.363
	UDSL	search	0.159	**0.632**	**0.532**	+0.422
WMDP-cyber	GradDiff	published	0.246	**0.255**	0.192	0
	UDSL	migrate	0.262	0.230	0.170	−0.022
	UDSL	search	**0.245**	**0.255**	0.193	+0.001
ConceptGuard	GradDiff	published	**0.000**	0.493	**0.493**	0
	UDSL	migrate	0.067	0.203	0.189	−0.304
	UDSL	search	**0.000**	0.464	0.464	−0.029
RWKU	GradDiff	published	**0.000**	0.004	0.004	0
	UDSL	migrate	0.033	0.042	0.040	+0.036
	UDSL	search	0.060	0.072	0.068	+0.064

Zero-shot migration is unreliable across datasets. On MUSE-News and WMDP-cyber, migration remains at or below GradDiff, while on MUSE-Books it improves substantially over the published baseline despite being obtained without additional search. These differences show that the discovered procedure itself does not provide a universal cross-dataset solution.

Independent UDSL search is more consistently useful than direct migration on the datasets where the search can escape the baseline trade-off. On MUSE-News and MUSE-Books, re-search improves the Score over GradDiff, while WMDP-cyber is essentially tied. On ConceptGuard, GradDiff already achieves zero Forget, and UDSL search does not recover its full utility. RWKU remains a low-utility regime for all compared methods.

The results therefore support a narrower claim: UDSL provides a reusable representation for dataset-specific search, but it does not guarantee an improvement on every benchmark under the same evaluation budget.

4.5 Why Does UDSL Search Work?

The main results establish that UDSL can enable stronger search, but do not by themselves identify which property of the representation is responsible. We therefore conduct two ablations. First, Table 4 removes different forms of compositional or semantic search while keeping the evaluation budget fixed. Second, Table 5 freezes one axis of the discovered program and searches the remaining axes.

Under the same 13-evaluation budget, removing composition, restricting search to hyperparameters, or searching over named trainers fails to leave the SimNPO parent envelope. Importantly, several failed variants still produce many unique candidates, indicating that proposal diversity alone does not explain the difference. Instead, the successful search requires a representation that allows new combinations of unlearning mechanisms to be expressed. Figure 4 makes this concrete with one real OpenEvolve step from each interface: the named-trainer arm rewrites a Python `get_unlearn_config()` dict and swaps `GradDiff` for `NPO` (Fitness $0.411 \rightarrow 0.195$), whereas the UDSL arm applies a single `subtree_replacement` on the forget signal, `abs(vs_ref)`→`softplus(vs_ref)`, and that child is the search winner. The search loop is the same; the writable object is not.

Table 4: Equal-budget representation ablation (13 evaluations). × marks a disabled UDSL ingredient. Mean / Uniq. / E2B measure whether each search arm climbs or resamples the SimNPO parent envelope (0.435).

Variant	Fit.↑	Mean	Uniq.	E2B	Δ
UDSL full	**0.442**	–	–	10	0
OpenEvolve named (50)	0.434	–	–	4	−0.008
× No composition	0.435	0.157	10	7	−0.007
× No semantic mutation	0.437	0.413	8	11	−0.005
× Random mutation	0.435	0.214	12	13	−0.007
× Human-designed seeds	0.435	0.252	9	3	−0.007
× Hyperparameters only	0.435	0.207	5	4	−0.007

Table 5 further examines the discovered program by freezing one semantic axis at a time while searching the remaining axes.

Table 5: Freeze one axis of the discovered UDSL program and search the other three for 13 evaluations. Rec. = a child reaches Fitness ≥ 0.441. Coll. = Fitness < 0.01. The additional Update-only experiment is discussed in the text.

Frozen	Fit.	Δ	Mean	Uniq.	Coll.	Rec.	E2R
none	0.442	0	–	–	–	yes	–
Signal	0.130	−0.313	0.037	11	2	no	never
Update	0.442	0	0.327	11	1	yes	1
Scope	**0.447**	+0.004	0.231	12	1	yes	1
Protect	0.443	+0.001	0.248	11	0	yes	1

Freezing Signal substantially restricts search: Fitness falls to 0.130 and no candidate recovers the original region within the evaluation budget. In contrast, freezing Update, Scope, or Protect allows the search to recover a near-original solution quickly. This suggests that, for the discovered procedure and under the current budget, the Signal component is particularly important, while the remaining axes provide additional degrees of freedom for local recovery. When Update is the only axis allowed to change, directly replacing the discovered $\mathrm{logsigmoid}$ update causes Fitness to fall to 0.035, further indicating that the usefulness of the axes depends on their composition rather than on any single operation considered in isolation.

Takeaway. The ablations point to composition as the key distinction between UDSL search and simpler alternatives. Coefficient tuning, named-method recombination, and proposal diversity alone remain within the parent envelope, whereas the full representation can express and discover a new composition. Appendix C shows the corresponding prompts. The absolute improvement over the named-trainer search is small and based on a single reported seed; we therefore interpret it as evidence about the search space rather than as a claim of a large performance margin.

5 Conclusion

We presented UDSL, a compositional representation for LLM-guided discovery of machine unlearning algorithms. By organizing the search space along Signal, Update, Scope, and Protect, the language lets an LLM mutate algorithmic structure while an interpreter keeps the trainer stack fixed. On TOFU-forget10, a 13-evaluation search reaches Fitness 0.442, above GradDiff (0.409). One-shot UDSL proposals can also beat GradDiff (Claude-Opus-4.7: 0.438), though independent redraws of the same prompt vary (Appendix D). A discovered program does not transfer zero-shot across forget ratios or datasets; independent search helps on MUSE and at 3B, but not uniformly. Ablations indicate that removing composition or the discovered forget signal prevents leaving the parent envelope.

This study is limited by a short UDSL search budget, high variance across LLM repeats, a proxy fitness that is not official TOFU Forget Quality, and an encoding gap between published SimNPO (0.093) and its UDSL parent (0.434). Broader evaluation across seeds, official metrics, and larger structured-search budgets is important future work.

AI USE STATEMENT

In this work, we used generative AI tools as a core component of the proposed method: large language models generate and mutate candidate unlearning programs within UDSL, and the resulting programs are compiled and evaluated automatically. We have not used generative AI tools to fabricate experimental results, figures, or citations. Generative AI tools were used to assist with manuscript formatting and migration into the ICLR LaTeX style. We have reviewed all AI-assisted text and formatting. Experimental numbers, claims, and algorithmic descriptions were checked against the original experimental records. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies approximate machine unlearning as a mechanism for reducing the influence of designated training data, which is relevant to privacy, copyright, and data-governance settings. Primary experiments use TOFU, which consists of synthetic question–answer pairs derived from fictitious author biographies and does not involve human subjects or personally identifiable information. Additional transfer experiments use MUSE, WMDP-cyber, ConceptGuard, and RWKU. WMDP contains hazardous-knowledge probes (Li et al., 2024); we use it only as a published unlearning benchmark and do not attempt to increase hazardous capability. The proposed search procedure can in principle be applied to other models and datasets; we caution that unlearning methods should not be treated as a complete privacy guarantee, especially in the approximate-unlearning setting studied here.

REPRODUCIBILITY STATEMENT

We specify the evaluation protocol in Section 4, including the TOFU setting, Llama-3.2-1B-Instruct backbone, optimizer, and the MU, FP, and Fitness metrics. The unlearning objective and UDSL axes are described in Sections 3.1 and 3.2. Quantitative comparisons are reported in Tables 1, 2, 3, 4, and 5. Cross-dataset plots of Table 3 are in Appendix B. A side-by-side OpenEvolve step (Python config vs. UDSL tree) is in Appendix C. The evolution trajectory, repeat proposals, and the discovered program are given in the appendix.

REFERENCES

Matthew Bowers, Theo X. Olausson, Lionel Wong, Gabriel Grand, Joshua B. Tenenbaum, Kevin Ellis, and Armando Solar-Lezama. Top-down synthesis for library learning. *Proceedings of the ACM on Programming Languages*, 7(POPL):1182–1213, 2023. doi: 10.1145/3571234.

Pengfei Cao, Chenhao Wang, Zhitao He, Hongbang Yuan, Jiachun Li, Yubo Chen, Kang Liu, Jun Zhao, et al. Rwku: Benchmarking real-world knowledge unlearning for large language models. *Advances in Neural Information Processing Systems*, 37:98213–98263, 2024.

Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*, pp. 463–480. IEEE, 2015. doi: 10.1109/SP.2015.35.

Yijiang River Dong, Hongzhou Lin, Mikhail Belkin, Ramon Huerta, and Ivan Vulić. UNDIAL: Self-distillation with adjusted logits for robust unlearning in large language models. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *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. 8827–8840, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.444. URL https://aclanthology.org/2025.naacl-long.444/.

Ronen Eldan and Mark Russinovich. Who's harry potter? approximate unlearning for LLMs. *arXiv preprint arXiv:2310.02238*, 2023.

Chongyu Fan, Jiancheng Liu, Licong Lin, Jinghan Jia, Ruiqi Zhang, Song Mei, and Sijia Liu. Simplicity prevails: Rethinking negative preference optimization for LLM unlearning. In *Advances in Neural Information Processing Systems*, 2025.

Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified data removal from machine learning models. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 3832–3842. PMLR, 2020.

Sahil Kale and Ian Harris. ConceptGuard: Benchmarking context-sensitive unlearning in large language models. *arXiv preprint arXiv:2608.20338*, 2026.

Nathaniel Li, Alexander Pan, Anjali Gopal, Summer Yue, Daniel Berrios, Alice Gatti, Justin D. Li, Anne-Kathrin Dombrowski, Shashwat Goel, Long Phan, Gabriel Mukobi, Nathan Helm-Burger, et al. The WMDP benchmark: Measuring and reducing malicious use with unlearning. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32201–32223. PMLR, 2024.

Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary Chase Lipton, and J. Zico Kolter. TOFU: A task of fictitious unlearning for LLMs. In *ICLR 2024 Workshops: DPFM*, 2024.

Rubens O. Moraes and Levi H. S. Lelis. Searching for programmatic policies in semantic spaces. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 5990–5998, 2024. doi: 10.24963/ijcai.2024/662.

Thanh Tam Nguyen, Thanh Trung Huynh, Zhao Ren, Phi Le Nguyen, Alan Wee-Chung Liew, Hongzhi Yin, and Quoc Viet Hung Nguyen. A survey of machine unlearning. *ACM Transactions on Intelligent Systems and Technology*, 2025. doi: 10.1145/3749987.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Weijia Shi, Jaechan Lee, Yangsibo Huang, Sadhika Malladi, Jieyu Zhao, Ari Holtzman, Daogao Liu, Luke Zettlemoyer, Noah A. Smith, and Chiyuan Zhang. MUSE: Machine unlearning six-way evaluation for language models. In *International Conference on Learning Representations*, 2025. URL `https://mlanthology.org/iclr/2025/shi2025iclr-muse/`.

Puning Yang, Qizhou Wang, Zhuo Huang, Tongliang Liu, Chengqi Zhang, and Bo Han. Exploring criteria of loss reweighting to enhance llm unlearning. *arXiv preprint arXiv:2505.11953*, 2025.

Yang Yang, Zining Zhong, Jindong Li, Jiemin Wu, Kaishen Yuan, Wenshuo Chen, Menglin Yang, and Yutao Yue. TurboEvolve: Towards fast and robust LLM-driven program evolution. *arXiv preprint arXiv:2604.18607*, 2026.

Yuanshun Yao, Xiaojun Xu, and Yang Liu. Large language model unlearning. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems*, 2024.

Xiaojian Yuan, Tianyu Pang, Chao Du, Kejiang Chen, Weiming Zhang, and Min Lin. A closer look at machine unlearning for large language models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 49483–49508, 2025.

Ruiqi Zhang, Licong Lin, Yu Bai, and Song Mei. Negative preference optimization: From catastrophic collapse to effective unlearning. *arXiv preprint arXiv:2404.05868*, 2024.

A Evolution trajectory on TOFU-forget10

Table 6 records 13 trained-ok evaluations from seed 5201. Three candidates improved the best-so-far logged fitness, which peaks at 0.439 at depth 3 (`p43_1739f05cbfee`). Table 1 reports Fitness 0.442 from MU and FP of the selected program under the main protocol; we treat that table as canonical and this log as the search trace.

Table 6: Fitness versus evolution step (seed 5201). Best-so-far is monotonic over trained-ok evaluations.

Eval	Candidate	Fitness	Best-so-far	Depth
1	`p43_4c33188a0094`	0.294	0.294	3
2	`p43_01e6478cb9b3`	0.365	0.365	3
3	`p43_aec695f63782`	0.294	0.365	3
4	`p43_573f103709f4`	0.319	0.365	3
5	`p43_e8cd95e12656`	0.078	0.365	3
6	`p43_04426728f8c0`	0.182	0.365	3
7	`p43_438c595768c5`	0.436	0.436	3
8	`p43_9acaa2258e9d`	0.000	0.436	3
9	`p43_a05cf182570e`	0.021	0.436	4
10	`p43_1739f05cbfee`	**0.439**	0.439	3
11	`p43_6dfae9695663`	0.020	0.439	4
12	`p43_1f41867215c8`	0.076	0.439	3
13	`p43_ada238099a3f`	0.276	0.439	3

B Cross-dataset plots

Figures 2 and 3 visualize Table 3. Search improves Score on MUSE-News and MUSE-Books; WMDP-cyber is essentially tied with GradDiff; ConceptGuard search matches Forget $= 0$ but loses utility; RWKU stays in a weak-utility regime. Migrate is not defined on TOFU-forget10, so that series is broken there. Δ_{GD} callouts on the Score panel mark the large MUSE-Books gain and the ConceptGuard drop.

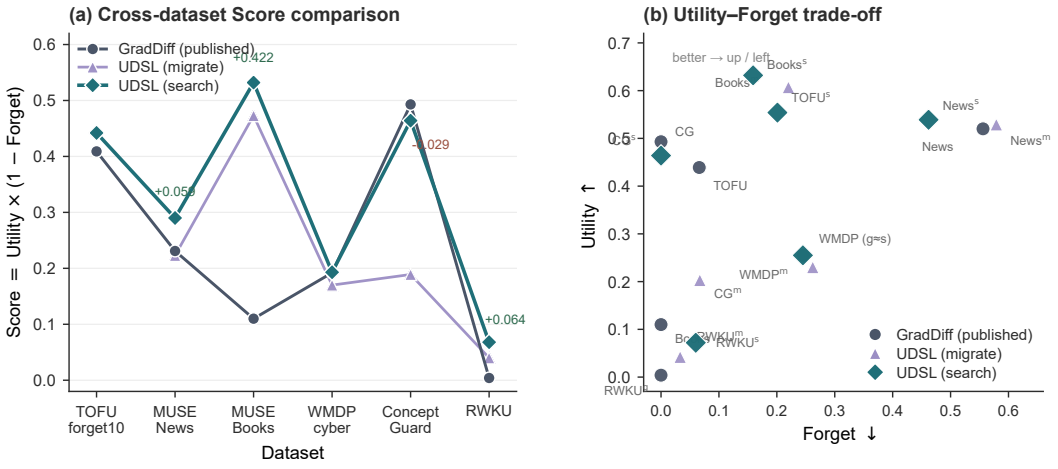

Figure 2: Cross-dataset read of Table 3. (a) Score $=$ Utility $\times$ (1 $-$ Forget) in table order for GradDiff, TOFU-program migration, and per-dataset search. (b) Forget–Utility scatter; better is up and left. Connecting lines in (a) compare datasets, not time.

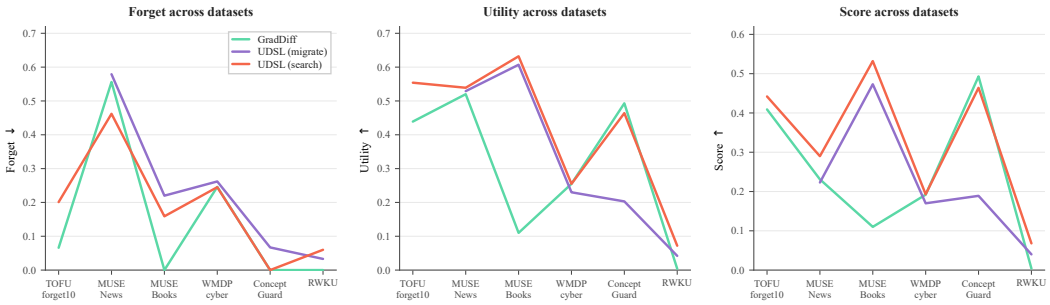

Figure 3: The three ingredients of Score in Table 3, aligned across datasets: Forget (↓), Utility (↑), and Score (↑). Search lowers Forget on News while recovering Utility on Books; migrate follows GradDiff more closely except on Books.

C What OpenEvolve rewrites

Table 4 reports that a 50-evaluation OpenEvolve search over named trainers stays at Fitness 0.434, while a 13-evaluation UDSL search reaches 0.442. Figure 4 shows one logged step from each arm, not a constructed example. On the Python side the LLM is asked to rewrite `get_unlearn_config()` and returns the same dictionary with `trainer` changed from `GradDiff` to `NPO`. On the UDSL side the mutator is asked for one structural edit and replaces the forget-signal node at path `b` (`abs`→`softplus` on `vs_ref_seq`); that program is the TOFU-forget10 winner. The figure is meant to make the ablation readable: diversity of Python rewrites does not expose a new forget-signal type.

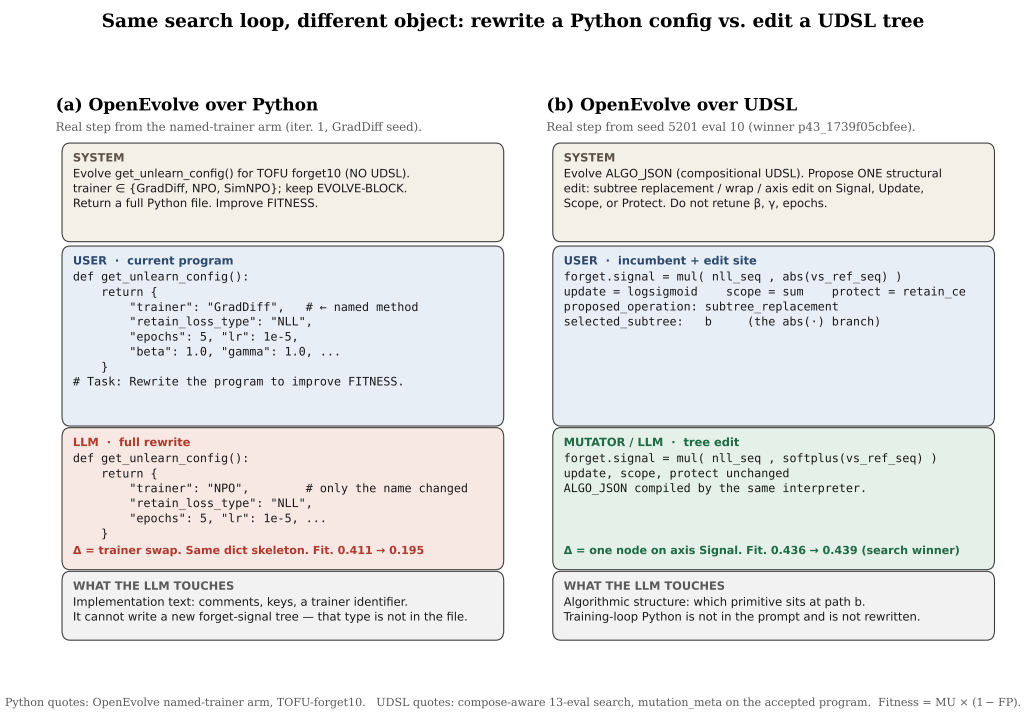

Figure 4: One OpenEvolve step under two representations. (a) Named-trainer Python arm: a full-file rewrite that swaps the trainer string (Fitness 0.411→0.195). (b) UDSL arm (seed 5201, eval 10): a `subtree_replacement` on Signal (Fitness 0.436→0.439). Quotes are from the stored prompts and `mutation_meta`.

D REPEAT LLM PROPOSALS

Table 7 is *not* a second main result. It records independent redraws of the same one-shot prompt. Means can disagree with Table 1 because that table reports one representative draw (including the stable Claude-Opus-4.6 re-run). The Python / GradDiff column is a different interface from HD trainer and is omitted from the main table.

Table 7: Repeat proposals (mean±std Fitness). n=3 except Claude rows (n=2). GPT-5.6 UDSL returned the same program on all three draws.

Model	Python / GradDiff	UDSL	HD one-shot
GPT-5.5	0.000±0.000	0.255±0.197	0.375
GPT-5.6	0.000±0.000	0.433±0.000	0.420
GPT-5.6-sol	0.149±0.257	0.446±0.008	–
Claude-Opus-4.6	0.000±0.000	0.215±0.264	0.308
Claude-Opus-4.7	0.000±0.000	0.000±0.000	0.387

E EVOLVED BEST PROGRAM

The best UDSL program uses a joint objective whose forget signal multiplies sequence NLL by a softplus-transformed reference contrast (`vs_ref_seq`), applies a `logsigmoid` update, aggregates with `sum`, and protects utility with `retain_ce`.

```
{
  "form": "joint",
  "forget": {
    "signal": {
      "op": "mul",
      "a": {"op": "nll_seq"},
      "b": {"op": "softplus", "x": {"op": "vs_ref_seq"}}
    },
    "update": {
      "op": "logsigmoid", "beta": 1.0, "delta": 0.0, "x": {"op": "$in"}
    },
    "scope": {"op": "sum", "x": {"op": "$in"}}
  },
  "protect": {"kind": "retain_ce"},
  "gamma": 1.0, "alpha": 1.0, "beta": 1.0, "delta": 0.0,
  "epochs": 5, "erase_epochs": 0, "repair_epochs": 0
}
```