LEARNING TO SEARCH FOR EXECUTABLE PATTERNMAKING FROM CERTIFIED PROGRAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent models generate sewing patterns from images or text, and program-based representations make these patterns executable. These systems produce a finished pattern and do not model how a patternmaker turns one pattern into another. In practice a new style is made by editing an existing pattern: lengthening the bodice, moving a dart, swapping a sleeve. Pattern archives record the results of such edits and none of the steps. We observe that the steps can be recovered by searching over editing actions until one pattern becomes the other. A deterministic compiler makes the recovered steps checkable. To this end, we present SemDraft, the first framework that learns to search for executable patternmaking programs from paired before-and-after sewing patterns without recorded editing steps. Specifically, a bounded search recovers a program of editing actions and certifies it only after three checks: it recompiles from scratch, every pair of its actions is classified as ordered or interchangeable, and an independent replay reaches the same pattern. The certified programs then train a model that ranks candidate actions inside the same search. On two frozen draws of 80 order-sensitive cases, with every other search component fixed, the learned ranker certifies 63 and 64 cases within 1,000 search executions, against at most 53 and 52 for non-learned rankers.

1 INTRODUCTION

Sewing patterns underlie digital garment creation, from virtual try-on to made-to-measure manufacturing. Recent systems generate complete patterns from images or text (Liu et al., 2023; He et al., 2024; Bian et al., 2025; Nakayama et al., 2025; Weng et al., 2026), and program-based representations make these patterns executable (Korosteleva & Sorkine-Hornung, 2023; Zhou et al., 2025). Patternmakers rarely draft a style from nothing. They edit an existing pattern by lengthening a bodice, adding and transferring darts, redistributing ease, or swapping a sleeve template. The expertise that makes garments fit is therefore stored only as its outcomes. We study this setting: given the pattern before the edits and the pattern after, recover an executable program of editing actions that connects them.

Recovering the editing actions is challenging for three reasons. First, the supervision is missing. Even when a base–target pair is available, industrial CAD archives keep no log of the operations that connect them. Second, a garment action is coupled. One action changes several panels, their seam partners and their symmetry groups at once, whereas the operations recovered by inverse CAD methods act on a single solid (Xu et al., 2021; Khan et al., 2024a; Dupont et al., 2024; Rukhovich et al., 2025). Third, order is ambiguous. Many orders of the same actions produce the same pattern, and some orders do not, because an attached sleeve template resolves its geometry against the host panels as they are at that moment. A single recovered sequence therefore leaves open which other orders of its actions are valid.

Our insight is that a deterministic compiler makes a recovered sequence checkable. Once a compiler executes the editing actions, a program that reaches the target can be recompiled from scratch. Each pair of its actions can then be classified as ordered or interchangeable, and an independent replay can confirm the result. We build SemDraft on this insight (Fig. 1). A drafting environment compiles nine semantic patternmaking actions to geometry. A bounded teacher search recovers programs and accepts only those that pass the three-stage certification. The certified programs train a learned action-selection prior that ranks the environment's legal candidates. Every training program is therefore executable and carries a certified dependency DAG, with the valid next actions at each

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.

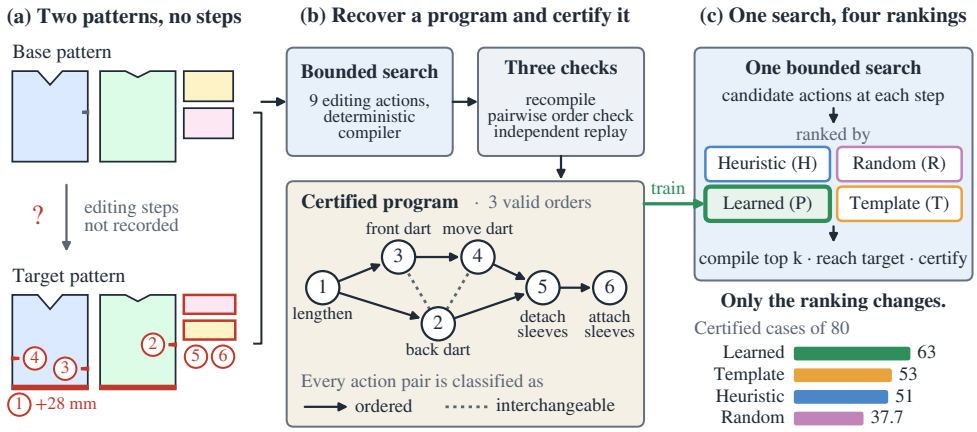


Figure 1: Overview of SemDraft on one case of the development corpus. (a) The input is a base pattern and a target pattern; the editing steps between them are not recorded. Red marks the changes, numbered by the step that makes them. (b) A bounded search over nine editing actions, executed by a deterministic compiler, recovers a program, and three checks certify it; the certificate classifies every pair of actions as ordered or interchangeable, which here admits three orders. (c) Certified programs train a model that ranks candidate actions (P). It is compared with the teacher heuristic (H), a random order (R) and a template lookup (T) inside one bounded search with the same candidates, beam, stopping rule, certifier and budget; bars: certified main-test cases of 80 at beam 4.

intermediate state. A unified bounded search then runs every ranking signal, learned or not, with the same candidates, beam, stopping rule, certifier and a shared execution budget.

We evaluate on a newly generated and frozen test set of 80 order-sensitive cases, each requiring a template swap whose landing depends on the host panels. Under matched conditions, the learned prior with a beam of four certifies 63 of the 80 cases, averaged over two training seeds. A lookup of the training corpus's action-type statistics certifies 53, the teacher's own ranking heuristic 51, and a random order 37.7. The comparisons are paired on the same cases, the bootstrap range of each gain lies above zero, and an independent draw of 80 further cases reproduces them. The teacher itself, run unchanged on the same cases with its full wall-clock budget, certifies 30. Of the 50 cases it misses, the prior certifies 34 and the teacher heuristic 21, each within 1,000 search executions.

In summary, our main contributions are:

- We present the first framework that learns to search for executable patternmaking programs from paired before-and-after sewing patterns without recorded editing steps. A bounded search over semantic patternmaking actions accepts a program only after a three-stage certification: recompilation, a pairwise order check and an independent replay. Every training program is therefore executable and carries its certified dependency DAG and the valid next actions at each of its intermediate states.
- We propose a learned action-selection prior and evaluate it under matched conditions. The prior ranks the environment's legal candidates inside one fixed bounded search. We compare it with the teacher's heuristic, a random order and a template lookup under the same candidates, beams, stopping rule, certifier and execution budgets.
- We establish a frozen evaluation with replication, distribution shift and teacher-missed solutions. The test set is freshly generated, de-duplicated and split before use, with an independent replication draw and two distribution-shift sets. On the cases the teacher itself misses within its wall-clock budget, we test whether the prior finds certified programs.

2 Related Work

Sewing-pattern generation and alteration. This line produces or modifies complete sewing patterns. GarmentCode and GarmentCodeData define parametric, executable pattern programs

and a made-to-measure dataset (Korosteleva & Sorkine-Hornung, 2023; Korosteleva et al., 2024). SewFormer, DressCode, ChatGarment, AIpparel, Design2GarmentCode and GarmentGPT generate or edit patterns from images, text or both (Liu et al., 2023; He et al., 2024; Bian et al., 2025; Nakayama et al., 2025; Zhou et al., 2025; Weng et al., 2026), and GarmentGPT replaces coordinate regression with discrete, compositional latent tokens. Digital garment alteration is the closest formulation to ours: it edits an existing pattern with tailoring operations and fits it to a new body shape (Eggler et al., 2024). Concurrently, TailorCoPilot records the operations that experts perform in a pattern-making tool as explicit sequences over structured pattern states (Sun et al., 2026). SemDraft recovers such sequences from the two patterns alone and learns to choose the next action, anchor and amount.

Recovering construction sequences and programs. Inverse CAD recovers how an object was built. With recorded histories, DeepCAD, SkexGen and Text2CAD model ordered command sequences (Wu et al., 2021; Xu et al., 2022; Khan et al., 2024b). From final geometry, Zone Graphs, PLAD, CAD-SIGNet, TransCAD and CAD-Recode infer sequences or code (Xu et al., 2021; Jones et al., 2022; Khan et al., 2024a; Dupont et al., 2024; Rukhovich et al., 2025), ExtrudeNet, SECAD-Net and SfmCAD learn sketch-and-extrude decompositions without them (Ren et al., 2022; Li et al., 2023; 2024), and cadrille adds reinforcement learning from programmatic feedback (Kolodiazhnyi et al., 2026). VPI-Edit learns local edits that move a visual program towards a target and evolves a population of programs with them (Jones et al., 2024). CAD-Assistant applies tool-augmented VLMs to CAD tasks (Mallis et al., 2025), and execution-guided synthesis trains a learner on programs found with an executor in the loop (Ellis et al., 2019; 2021). These methods reconstruct a single shape from its geometry. A garment action changes several panels and their seams, and many orders of the same actions reach the same pattern. SemDraft recovers a certified partial order of parameterised actions over coupled panels, with one executor for training and evaluation.

Learning from search and verification. A common pattern trains a model on the outputs of a stronger search and then uses it to guide a cheaper search. Expert iteration, AlphaGo Zero, pseudo-label program induction such as PLAD, and library learning in DreamCoder follow this pattern (Anthony et al., 2017; Silver et al., 2017; Jones et al., 2022; Ellis et al., 2021). In program synthesis, learned models order a fixed enumeration or deduction (Balog et al., 2017; Kalyan et al., 2018) or act as the search policy (Shi et al., 2022), and a learned policy can improve on a suboptimal reference (Chang et al., 2015). Verifier-filtered self-training is common in reasoning tasks (Cobbe et al., 2021; Singh et al., 2024). When several next actions are equally correct, dynamic oracles supply the set of optimal transitions (Goldberg & Nivre, 2013), and training can maximise their total probability (Ballesteros et al., 2016), the objective we use to train the prior. Deordering a sequential plan into a partial order is classical in planning (Muise et al., 2016), and the equivalence of interleavings under an independence relation is the subject of trace theory (Mazurkiewicz, 1987). A concurrent system certifies programs recovered from execution video with a checker over such an independence quotient (Si & Zhang, 2026). SemDraft decides independence by executing both orders in a geometric compiler and keeps a program only after all three checks succeed. It trains the prior to place its probability mass on the valid next actions that the certificates provide.

3 METHOD

Given a base pattern S_0 and a public target, SemDraft searches for an executable program that connects them. A compiler executes the actions, a teacher search recovers and certifies programs, and a learned prior ranks candidates inside the same search.

3.1 DRAFTING ENVIRONMENT: SEMANTIC ACTIONS OVER A DETERMINISTIC COMPILER

State and actions. A pattern state S has panels (closed polygons with semantically labelled edges), seams, components such as a bodice or sleeve pair, and internal darts as separate entities. Edits use the vocabulary of patternmaking. The ontology has nine semantic actions: introduce internal dart, transfer internal dart, redistribute component ease, true a straight seam pair, adjust component length, apply axis slash-and-spread, insert style line, and attach or detach a finite template component. Each action names the panels, boundaries, darts or components it touches, and carries continuous parameters, given in millimetres or in normalised units.

Compiler. One semantic action changes several panels, their seam partners and their symmetry groups together, so actions are executed by a compiler. The compiler lowers every action to a fixed set of geometric primitives (lengthen/shorten, widen/narrow, reshape boundary, slash-spread, split, merge, attach, detach). Execution is deterministic and transactional, with stable entity identities and full lineage. A transition that breaks closure, self-intersects, or violates seam compatibility or dart validity is rejected. Determinism makes a recovered program checkable, because the same program always reproduces the same state and artefact hash. Execution is also order-sensitive: attaching a template component such as a sleeve resolves its geometry against the current host panels, so the same swap yields different sleeves before and after a length adjustment (Fig. 2). The compiler is fixed throughout the paper, and learning enters only through the ranking of candidates.

Public endpoint criterion. The teacher and the prior observe a public target: panel contours, boundary semantics, dart parameters and component tags. The generating program stays hidden. A state is *tight* with respect to the target when three conditions hold. Eight exact structure predicates agree with the target: components, panels, boundary semantics, topology, seam graph, darts, style-line tags and template tags. A complete correspondence exists between the entities of the state and those of the target. Every panel contour lies within fixed public thresholds of its matched target contour: Chamfer ≤ 0.005, Hausdorff ≤ 0.020 and aligned RMS ≤ 0.010. Each distance is normalised by the centroid and bounding-box diagonal of the target contour and measured on 512 samples per contour.

3.2 Teacher: bounded search and certification

Search. Archives provide no programs, so SemDraft first recovers them with a teacher, a bounded search whose acceptance test is a three-stage certification. Given S_0 and the public target, the teacher runs a beam search over programs of bounded depth. At each prefix, nine proposal providers, one per action type, enumerate candidate actions that are consistent with the state–target correspondence. A dart present in the target and absent in the state yields a dart proposal. Every candidate is compiled and evaluated with the public metric. Prefixes are ranked lexicographically by structural mismatches, then contour distance, then program cost, and a beam of B prefixes survives per depth. The search is bounded in wall clock and in engine executions (Appendix A). Two mechanisms serve template swaps, which every case of the hardest benchmark tier requires: a one-step lookahead from each proposed detach to its re-attach variants, and a certification reserve of wall-clock time. Both mechanisms were developed using an aggregate census of the hidden programs of the development corpus and form part of the teacher's reference configuration. This configuration is run separately and unchanged; the unified search uses its ranking rule within a shared search loop.

Certification. A prefix whose endpoint is tight is a terminal candidate. It becomes *certified* when three independent checks succeed. The first check recompiles the whole program from S_0, which must reproduce the terminal state and its artefact hash. The second, a pairwise order check, gives every pair of actions one verdict. A pair whose static read/write effects conflict is recorded as a required precedence without execution. A pair that becomes adjacent in some explored order is executed in both orders from the shared prefix, and the artefact hashes are compared. Equal hashes give a commuting pair, two valid and different results give a required precedence, and a side that cannot be materialised leaves the pair undecided. Each pair is decided once and cached, and a pair that never becomes adjacent keeps the recovered order as a precedence implied by those already established. The check is complete when the exploration of commuting swaps closes with no undecided pair. The certifier then enumerates the reachable linear orders, materialises every downward-closed subset of the resulting dependency DAG as an executable intermediate state, and applies a minimality rule: no proper downward-closed subset may already be tight. The third check, an independent replay, runs the compiled low-level program in an oracle-isolated evaluator, verifies lineage completeness and locality, and confirms that the endpoint is tight under the hidden full state. Appendix E states what a certificate establishes and where it is conservative. A certificate is the unit of supervision: it yields the program, its dependency DAG and, for every materialised intermediate state, the set of *valid next actions* (nodes whose required predecessors are complete) and whether stopping is allowed there.

3.3 LEARNED PRIOR: A FACTORISED MODEL TRAINED ON CERTIFIED PROGRAMS

Model. The teacher ranks a candidate only after compiling it; the prior ranks candidates before compilation. It observes what the teacher observes: the panels, darts and components of the current state, the panels and darts of the public target, the state–target correspondence, the public per-panel metrics, and the actions executed so far. Each panel, target panel, component, dart and past action becomes a token with geometric statistics, and a global token carries the aggregate metrics and mismatch counts (Appendix D). A four-layer Transformer encoder (Vaswani et al., 2017) with hidden dimension 128 and four attention heads feeds three output heads. A type head scores the nine action types and STOP, an anchor head points to a panel or component token given the type, and a parameter head regresses millimetre amounts and classifies discrete slots.

Training. For every materialised state s of a certified program, the positive set $\mathcal{P}(s)$ holds all valid next actions, together with STOP where stopping is allowed. With $p(a \mid s)$ the prior's probability of action a in s, the loss is

$$\mathcal{L}(s) = -\log \sum_{a \in \mathcal{P}(s)} p(a \mid s), \tag{1}$$

applied to the type and to the anchor given each positive type, plus regression and classification losses on the parameters of the positives. We call this *DAG supervision*. The ablation *linear imitation* uses the same model and data and restricts $\mathcal{P}(s)$ in Eq. (1) to the next action of the terminal's linear order. This is how behaviour cloning would treat a recovered sequence.

3.4 UNIFIED BOUNDED SEARCH: ONE SEARCH, FOUR RANKING SIGNALS

Search loop. To measure what a ranking signal contributes, every other component of the search is held fixed. At each prefix the nine providers enumerate the legal candidates as for the teacher, up to eight per provider. A ranker orders the candidates, and they are compiled in that order until k of them compile ($k \in \{1, 4\}$). The k best children over all expanded prefixes form the next beam, where a child is ranked by the signal that ordered it. The search ends when a prefix is tight, when the depth limit of eight is reached, or when the execution budget is exhausted. All ranking signals share the public tight stopping rule, isolating the contribution of candidate ranking. The budget counts the engine executions spent in the search, which we call search executions. Each compilation is charged the executions it uses, and the budget is set as a fraction of the teacher's per-tier execution cap. Certification is charged separately, with the tier's cap as its limit.

Ranking signals. The heuristic H is the teacher's own ranking rule. It compiles every candidate and ranks the compiled states by the public metric key, as the teacher does inside its beam. The random order R is averaged over three seeds. The template lookup T orders candidates by the empirical probability of their action type given the tier and the step index in the certified training programs, and uses no geometry. The learned prior P is trained with DAG supervision or linear imitation. It scores a candidate by the log-probabilities of its action type and of its anchor given that type, plus a term that rewards amounts close to those the prior predicts (Eq. (2) in Appendix D). A child inherits the accumulated score of its prefix. H pays for its ranking information by compiling all candidates at every step, whereas R, T and P compile only the candidates that they select. A run achieves *certified success* at budget b if its first tight program is found within b search executions and passes the three-stage certification. The choices of a run are independent of the budget, so one run per condition yields the whole success-versus-budget curve.

4 BENCHMARKS: A DEVELOPMENT CORPUS AND FROZEN EVALUATION SETS

Generator. All corpora come from one generator. Each case starts from the neutral base pattern of its family (tee, skirt, pants or dress, each with a base dart and a detachable template component) and applies a hidden program to obtain the target. Programs are templated: the generator fixes the action-type order for each tier and depth, and a hash of the corpus label and the case slot picks the anchors and amounts from small value sets. Tier A programs have depth 2–3, Tier B 4–5 and Tier C 6–8. Every Tier-C program contains a detach/attach template swap, so the position of the swap relative to the length, dart and ease actions is part of the problem. Templating has two consequences.

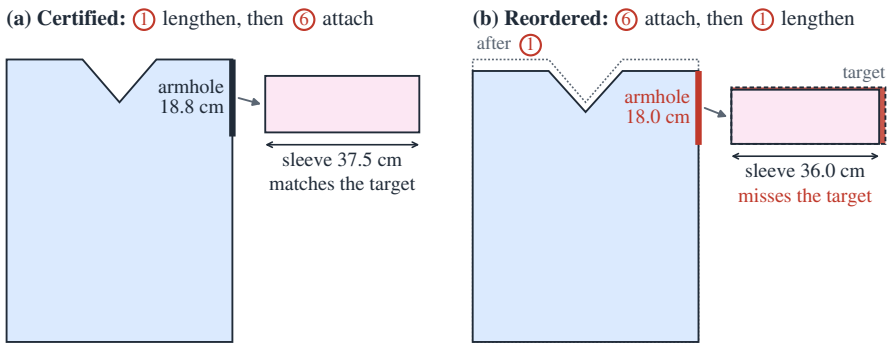


Figure 2: In the case of Fig. 1, the sleeve attachment has to follow the length adjustment, because the compiler sets a sleeve's width to the combined length of the front and back armholes it is sewn to. Compiled front bodice and right sleeve at one scale, without the darts, which end up identical in both orders; solid outlines show the state when the sleeve is attached. (a) In the certified order the sleeve matches the target. (b) Attached first, the sleeve is sized from the 18.0 cm armhole and keeps that size after the bodice is lengthened (dotted), so the program misses the target (Appendix E).

The action-type sequence of a case is largely determined by its tier and depth, and the template lookup T tests how far this knowledge alone goes. What remains for the search is the anchor and amount of each action and, at Tier C, an order in which the swap follows the host changes it depends on. At run time the teacher, the prior and the evaluation scripts receive public fields only, and a code-level guard rejects any oracle field that reaches the search interface.

Development corpus. Our development corpus contains 300 cases, 25 per tier and family; 5 cases per tier and family were used to develop the teacher and 20 are held out. It is the benchmark on which the teacher was developed, and it serves three purposes here: its certified programs train the prior, it supports protocol development, and it provides diagnostics. Teacher budgets are fixed per tier.

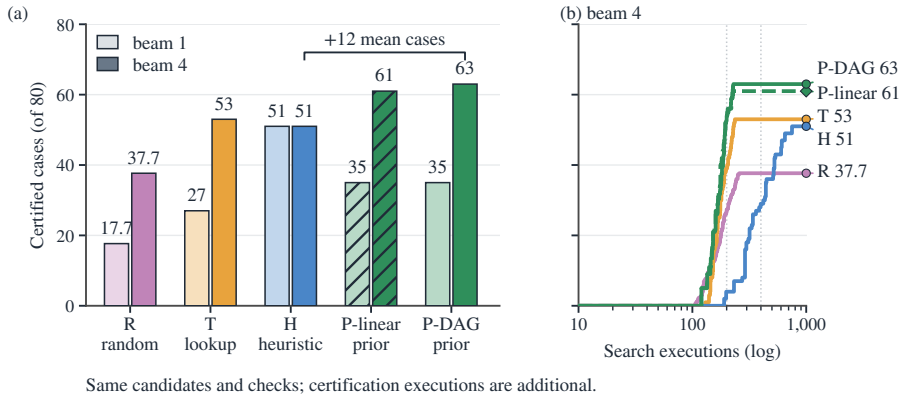
Frozen evaluation sets. For the main comparison we generated a new corpus of 300 cases with the same generator under a new corpus label. It was frozen before any ranker ran on it, as recorded in Appendix A. Five of its 300 targets overlap with the development corpus; none belongs to the declared splits below. Three splits were declared before use. 24 calibration cases, two per tier and family, fix the budgets and beam widths. The main test is the 80 held-out Tier-C cases (20 per family), and it carries the primary comparison. An easy reference of 20 held-out Tier-B cases shows the regime where the candidate generators suffice. Three further sets test robustness: an independent replication draw of 80 Tier-C cases, and a parameter shift and a template shift of 20 cases each. They were added after the main results had been read and fixed before use. The teacher's reference configuration, unchanged from the development-corpus runs, is also run on the whole main test, and the cases it misses within its wall-clock budget define the evaluation of Sec. 5.5. Every experiment follows a written protocol whose budgets, seeds and gates were fixed before its results were read (Nosek et al., 2018), and Appendix A lists the hardware and software and logs all deviations.

5 Experiments

5.1 Supervision: certified programs and the trained prior

Teacher on the development corpus. The teacher certifies 77, 70 and 57 of the 80 cases of Tiers A, B and C (Table 8, Fig. 6). On Tier C every tight case is also certified, and certified cases use a median of 1,442 engine executions and 1,315 s of wall clock. Most of the 23 non-certified Tier-C cases show *beam collapse*. The beam fills with reorderings of the same one or two sets of actions, all reaching the same state, and loses the branch that leads to the target (Table 7).

Training data and teacher-forced accuracy. The training data are the teacher's certified programs on the development corpus, 1,510 certified terminals over 185 cases: 77 in Tier A, 51 in Tier B and



Same candidates and checks; certification executions are additional.

Figure 3: Matched-condition comparison on the 80 Tier-C main-test cases. (a) Certified cases per ranking signal at beam 1 (light) and beam 4 (dark) within 1,000 search executions, the 5% budget; hatching marks the linear-imitation variant of the prior. (b) Certified cases against search executions at beam 4, read off the same runs; dotted lines mark 200 and 400 executions (1% and 2% of the cap).

57 in Tier C. They yield 2,846 materialised states after de-duplication by (case, completed action set), each with its set of valid next actions. A validation split of 20% of the cases per tier contributes no training states. Teacher-forced on this split, the prior predicts a valid next action type 92% of the time against a 49% majority baseline, and the anchor entity 99% of the time. Its millimetre amounts halve the error of a mean predictor. Linear imitation is indistinguishable on these metrics, as expected when the extra positives rarely change the predicted argmax (Table 10).

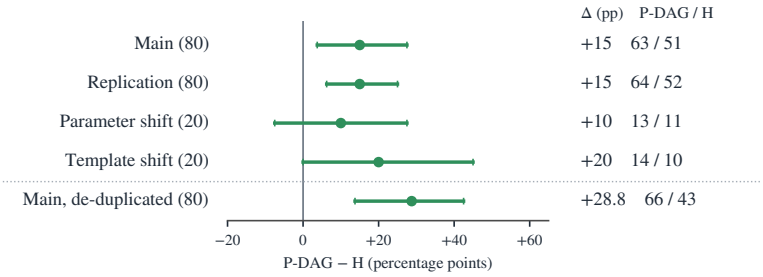
5.2 Matched-condition comparison on the frozen evaluation set

The primary endpoint, fixed before the runs, is certified success within 1,000 search executions (5% of the Tier-C cap) with a beam of four, and beam 1 is secondary. Two rankers are compared on the same cases, and Table 3 gives the case-bootstrap range of every gain.

Main result. With a beam of four, the learned prior certifies more cases than any non-learned ranking signal (Fig. 3 and Table 1). P certifies 63 of the 80 main-test cases, the mean of its two training seeds, 61 and 65. The template lookup T certifies 53, the teacher heuristic H 51 and the random order R 37.7, and P with linear imitation certifies 61. The gain over H is 12 cases, or 15 percentage points, and meets the criterion fixed before the runs: at least ten points, with a bootstrap range above zero. The gains over T and R, 10 and 25 cases, also have ranges above zero. T knows the action-type statistics of every tier and step and certifies 27 cases at beam 1 and 53 at beam 4, most of them at depths 6 and 7. Twelve cases are certified by P with both seeds and not by T, and two by T alone. At depth 8 T certifies 5 of 24 cases, against 10 for P and for H (Table 4).

Beam width and search cost. H reaches its 51 cases with a median of 103 search executions at beam 1 and 351 at beam 4, and the wider beam adds no case. At beam 1 P commits to one candidate per step and certifies 35 cases, fewer than H, because the metric feedback of H corrects a wrong first choice. On the 44 cases that both certify at beam 4, P needs a median of 160 search executions and H 325, and the median per-case ratio is 1.9. On the 49 such cases of the replication draw it is 1.8. A run ends when its beam is exhausted, after about 220 executions for P. P, R and T therefore gain no cases after 400 executions. H at beam 4 keeps rising, from 29 cases at 400 executions to 51 at 1,000, and no ranker changes between 1,000 and 4,000 (Fig. 3b).

Tight and certified. Eight of P's tight endpoints at beam 4, nine of T's at beam 1 and one of H's fail certification. All are eight-step programs that reach the target with one redundant step, and the certifier's minimality rule rejects them (Appendix E). On the easy reference every signal is tight on all 20 cases at beam 1, and 16 to 20 certify, so the differences belong to the order-sensitive tier.



All rows: beam 4, 1,000 search executions.

Figure 4: Paired gain of the prior (P-DAG) over the teacher heuristic (H) at beam 4 within 1,000 search executions, in percentage points; bars show the case-bootstrap range of each gain. The last row applies beam de-duplication to both. The right column gives the gain and both counts.

5.3 Where the gain comes from

Against H, the gain comes from tee and dress cases at depth 7. Table 4 splits the main test by family and target depth. At depth 6 both signals certify almost every case: P all 28 and H 25. At depth 7 P certifies all seven tee cases and all seven dress cases, and H one of each; both certify the seven skirt cases. At depth 8 the two are level at 10 of 24, and this depth remains open. Pants is the family on which H is stronger, with 18 of 20 cases against 11. P misses six pants cases at depth 8, where its beam is exhausted before a tight state, and three at depth 7 that reach the target with one redundant step. Of H's 29 misses, 27 are tee and dress cases, the families in which the teacher's beam fills with reorderings of the same actions on the development corpus.

Supervision form and beam width. DAG supervision and linear imitation are level: 35 against 35 at beam 1, and 63 against 61 at beam 4, within the spread of the two seeds, 61 and 65. On this templated corpus the sets of valid next actions add little to the single linear order, and the certificate serves as the acceptance test and as the filter for training programs. The beam width matters more: it raises P from 35 to 63, T from 27 to 53 and R from 17.7 to 37.7, and leaves H unchanged at 51.

5.4 Robustness: a second draw, shifted inputs and beam de-duplication

Replication. The result reproduces on a second draw of 80 cases, run with the same trained prior and protocol (Fig. 4 and Table 2). P certifies 64 cases at beam 4 with both seeds and with linear imitation, against 52 for H, 51 for T and 37.3 for R. The gain over H is again 12 cases, and the gain over T is 13 cases. Both gains have bootstrap ranges above zero, as does the gain over H of 24 cases on the 160 cases of both draws. At beam 1 the main-test order repeats: H certifies 52, P 46 and T 33.

Distribution shift within the ontology. The learned prior certifies the most cases under both controlled shifts. The *parameter-range shift* keeps the templates and moves every parameter value set to unseen values. The *template shift* keeps the parameter values and changes the action composition and dependency structure at all three depths. Both are evaluated without tuning. At beam 4 P certifies 13 cases under the parameter shift and 14 under the template shift, against 11 and 10 for H, 9 and 10 for T, and 9.7 and 9.3 for R. With twenty cases per set, the bootstrap ranges of the gains over H, 2 and 4 cases, include zero, and those of the gains over T, 4 cases on each set, lie above zero. Under shift the certified rate moves from 79% to 65–70% for P, from 64% to 50–55% for H and from 66% to 45–50% for T. That P loses the least argues against template memorisation.

Beam de-duplication. Removing the heuristic's duplicate beam slots widens the gap. Prefixes that reach the same full state through the same actions in a different order cause the teacher's beam collapse, and they can be merged to free beam slots. We added this merging to the unified search for H and for P at beam 4 after the main results had been read, and fixed the check before its run. P gains three cases, from 63 to 66 with seeds 65 and 67, and linear imitation rises from 61 to 64. H loses eight cases, from 51 to 43. H's beam held a median of 12 merged states per case against 2 for P, so

the merging mostly changes H's search. With merging on both sides P certifies 23 more cases than H, so the learned gain is not explained by the beam slots that H wastes on reorderings.

5.5 Beyond the teacher's reference configuration

The prior certifies programs that the teacher's reference configuration does not find within its wall-clock budget. It was run on all 80 main-test cases with its configuration and analysis rule fixed in advance. The configuration certifies 30 cases (Table 9). Of the other 50, 5 reach a tight endpoint that fails certification, 27 exhaust the search envelope, and 18 are stopped while writing their result after the search had ended. All 27 envelope cases stop at the wall-clock limit, far below the execution cap (median 1,482 of 20,000 executions). On the development corpus it certifies 57 of 80; the drop to 30 has the same main mechanism, beam collapse. On the 50 missed cases, P at beam 4 certifies 34 with both seeds within 1,000 executions, T 29, H 21 and R 14. On the 32 cases not stopped while writing their result, the counts are 25, 20 and 15 for P, T and H. Seventeen cases are certified by P and not by H, and four the other way; all five programs that the reference configuration reached without certifying are certified by both. The advantage is at depths 6 and 7, where P certifies 11 of 11 and 18 of 18 cases against 8 and 6 for H. At depth 8 both are weak, with 5 and 7 of 21.

6 Limitations

The development corpus and the frozen evaluation set are controlled, synthetic identifiability benchmarks, not evidence of industrial generalisation. Their hidden programs are drawn from the same nine-action ontology the search uses, the programs are templated per tier and depth, and the geometry is polygonal. The teacher's reference configuration has two identified weaknesses: its beam collapses onto reorderings of the same actions, and its certification reserve is too small for programs with many commuting actions. We left it unchanged so that the beyond-teacher comparison is against one fixed configuration. The prior was trained and evaluated on corpora from one generator. The shift sets vary parameters and templates but not the generator, families or ontology, and with twenty cases each their bootstrap ranges are wide; we claim no transfer to other generators. Certification checks executability and order relations against the compiler; it does not show that a professional would draft this way.

7 Conclusion

Executable patternmaking programs can be recovered from before-and-after sewing patterns and certified against a deterministic compiler. Trained on these certified programs, a learned prior raises certified success in one fixed bounded search from 51 cases with the teacher's heuristic to 63 of 80 order-sensitive cases. It certifies more cases than every non-learned ranking on both frozen draws. On the 50 cases that the teacher's reference configuration misses within its wall-clock budget, it certifies 34, against 21 for the heuristic. Certified programs thus turn pattern archives, which keep the results of edits but not the steps, into supervision for search.

AI use statement

We used generative AI tools to assist with manuscript writing and language polishing. All AI-assisted text was carefully reviewed and revised by the authors. LLM-generated code was verified and tested for correctness by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

Reproducibility statement

Section 3 specifies the drafting environment, the teacher search and its certification, the learned prior and the unified search. Section 4 and Appendix A give the corpus construction, the frozen splits, all budgets and seeds, the hardware and software, and every deviation from the protocol. Appendix D gives the model and training details, and Appendix E states what a certificate establishes.

REFERENCES

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. Training with exploration improves a greedy stack LSTM parser. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2005–2010, 2016.

Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. DeepCoder: Learning to write programs. In *International Conference on Learning Representations (ICLR)*, 2017.

Siyuan Bian, Chenghao Xu, Yuliang Xiu, Artur Grigorev, Zhen Liu, Cewu Lu, Michael J. Black, and Yao Feng. ChatGarment: Garment estimation, generation and editing via large Language models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé III, and John Langford. Learning to search better than your teacher. In *International Conference on Machine Learning (ICML)*, pp. 2058–2066, 2015.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

Elona Dupont, Kseniya Cherenkova, Dimitrios Mallis, Gleb Gusev, Anis Kacem, and Djamila Aouada. TransCAD: A hierarchical transformer for CAD sequence inference from Point Clouds. In *European Conference on Computer Vision (ECCV)*, 2024.

Anna Maria Eggler, Raphael Falque, Mark Liu, Teresa Vidal-Calleja, Olga Sorkine-Hornung, and Nico Pietroni. Digital garment alteration. *Computer Graphics Forum (Proc. Pacific Graphics)*, 43 (7), 2024.

Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a REPL. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Bootstrapping inductive program synthesis with Wake-Sleep Library Learning. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2021.

Yoav Goldberg and Joakim Nivre. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414, 2013.

Kai He, Kaixin Yao, Qixuan Zhang, Jingyi Yu, Lingjie Liu, and Lan Xu. DressCode: Autoregressively sewing and generating garments from text guidance. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 43(4), 2024.

R. Kenny Jones, Homer Walke, and Daniel Ritchie. PLAD: Learning to infer shape programs with pseudo-labels and approximate distributions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

R. Kenny Jones, Renhao Zhang, Aditya Ganeshan, and Daniel Ritchie. Learning to edit visual programs with self-supervision. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024.

Ashwin Kalyan, Abhishek Mohta, Oleksandr Polozov, Dhruv Batra, Prateek Jain, and Sumit Gulwani. Neural-guided deductive search for real-time program synthesis from examples. In *International Conference on Learning Representations (ICLR)*, 2018.

Mohammad Sadil Khan, Elona Dupont, Sk Aziz Ali, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-SIGNet: CAD Language inference from Point Clouds using layer-wise Sketch Instance Guided Attention. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024a.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2CAD: Generating sequential CAD designs from beginner-to-expert level text prompts. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024b.

Maksim Kolodiazhnyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. cadrille: Multi-modal CAD reconstruction with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

Maria Korosteleva and Olga Sorkine-Hornung. GarmentCode: Programming parametric sewing patterns. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 42(6), 2023.

Maria Korosteleva, Timur Levent Kesdogan, Fabian Kemper, Stephan Wenninger, Jasmin Koller, Yuhan Zhang, Mario Botsch, and Olga Sorkine-Hornung. GarmentCodeData: A dataset of 3D Made-to-Measure garments with sewing patterns. In *European Conference on Computer Vision (ECCV)*, 2024.

Pu Li, Jianwei Guo, Xiaopeng Zhang, and Dong-Ming Yan. SECAD-Net: Self-supervised CAD reconstruction by learning Sketch-Extrude operations. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Pu Li, Jianwei Guo, Huibin Li, Bedrich Benes, and Dong-Ming Yan. SfmCAD: Unsupervised CAD reconstruction by learning sketch-based feature modeling operations. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

Lijuan Liu, Xiangyu Xu, Zhijie Lin, Jiabin Liang, and Shuicheng Yan. Towards garment sewing pattern reconstruction from a single image. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 42(6), 2023.

Dimitrios Mallis, Ahmet Serdar Karadeniz, Sebastian Cavada, Danila Rukhovich, Niki Maria Foteinopoulou, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-Assistant: Tool-augmented VLLMs as generic CAD task solvers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Antoni Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency (Advances in Petri Nets 1986, Part II)*, volume 255 of *Lecture Notes in Computer Science*, pp. 278–324. Springer, 1987.

Christian Muise, J. Christopher Beck, and Sheila A. McIlraith. Optimal partial-order plan relaxation via MaxSAT. *Journal of Artificial Intelligence Research*, 57:113–149, 2016.

Kiyohiro Nakayama, Jan Ackermann, Timur Levent Kesdogan, Yang Zheng, Maria Korosteleva, Olga Sorkine-Hornung, Leonidas Guibas, Guandao Yang, and Gordon Wetzstein. AIpparel: A multimodal foundation model for digital garments. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.

Brian A. Nosek, Charles R. Ebersole, Alexander C. DeHaven, and David T. Mellor. The preregistration revolution. *Proceedings of the National Academy of Sciences*, 115(11):2600–2606, 2018.

Daxuan Ren, Jianmin Zheng, Jianfei Cai, Jiatong Li, and Junzhe Zhang. ExtrudeNet: Unsupervised inverse Sketch-and-Extrude for shape parsing. In *European Conference on Computer Vision (ECCV)*, 2022.

Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-Recode: Reverse engineering CAD code from Point Clouds. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Seamly2D contributors. Seamly2D: Open-source pattern design software. https://seamly.io, 2026.

Kensen Shi, Hanjun Dai, Kevin Ellis, and Charles Sutton. CrossBeam: Learning to search in bottom-up program synthesis. In *International Conference on Learning Representations (ICLR)*, 2022.

Yuan Si and Jialu Zhang. Checked program recovery from execution video: A sound oracle for untrusted generators. *arXiv preprint arXiv:2607.00635*, 2026.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, et al. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024.

Yuexin Sun, Zhaohui Wang, Ruiyang Liu, Demian Kong, Qian He, Gaofeng He, and Huamin Wang. TailorCoPilot: Enabling agentic pattern making with version-controlled state tracking. *arXiv preprint arXiv:2608.25462*, 2026. To appear at ACM UIST 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Fangsheng Weng, Junhao Chen, Xiang Li, Jie Qin, Hanzhong Guo, Shaochun Hao, and Xiaoguang Han. GarmentGPT: Compositional garment pattern generation via discrete latent tokenization. In *International Conference on Learning Representations (ICLR)*, 2026.

Rundi Wu, Chang Xiao, and Changxi Zheng. DeepCAD: A deep generative network for computer-aided design models. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Xiang Xu, Karl D. D. Willis, Joseph G. Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. SkexGen: Autoregressive generation of CAD construction sequences with disentangled codebooks. In *International Conference on Machine Learning (ICML)*, 2022.

Xianghao Xu, Wenzhe Peng, Chin-Yi Cheng, Karl D. D. Willis, and Daniel Ritchie. Inferring CAD modeling sequences using Zone Graphs. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Feng Zhou, Ruiyang Liu, Chen Liu, Gaofeng He, Yong-Lu Li, Xiaogang Jin, and Huamin Wang. Design2GarmentCode: Turning design concepts to tangible garments through program synthesis. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025.