FHEVOLVE: CONSTRAINT-GUIDED EVOLUTION OF EFFICIENT FHE PROGRAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Fully homomorphic encryption (FHE) enables computation on encrypted data, but efficient implementations require coordinated choices of approximation methods, data layouts, multiplication depth, and rotation keys. General-purpose evolutionary code search can spend evaluations on infeasible candidates or repeatedly refine one algorithmic approach. We introduce **FHEvolve**, a framework that guides evolutionary optimization with reusable FHE knowledge. Given a workload specification, cryptographic configuration, and initial seed programs, FHEvolve extracts constraints, filters a library of algorithm strategies, preserves different strategy families during search, and translates execution failures into repair guidance. Across eight workloads spanning CKKS and BFV, FHEvolve improves the combined accuracy–latency score over vanilla OpenEvolve on several workloads while obtaining similar scores on others. On Softmax, encrypted evaluation time decreases from 8.48 to 7.11 seconds at unchanged measured accuracy. On encrypted fraud classification, a candidate selected under an additional probability-error constraint achieves a $1.3\times$ speedup and preserves agreement on 100 fresh examples; a faster candidate selected under binary agreement alone generalizes less well, highlighting the importance of the optimization objective. Against three FHE compilers (Orion, HEIR, CHEHAB), FHEvolve is the only system that produces a within-budget program for all eight workloads without a human-written algorithm, although a hand-tuned compiler is more accurate on two of them. These results support constraint-guided search as a practical approach to improving FHE programs, with benefits that depend on the workload and evaluation requirements.

1 INTRODUCTION

Fully homomorphic encryption (FHE) (Gentry, 2009; Cheon et al., 2017) lets a server evaluate functions on encrypted data without seeing the plaintext. It underpins encrypted ML inference (Hesamifard et al., 2018; Gilad-Bachrach et al., 2016), confidential smart contracts (Madathil & Scafuro, 2023; Z1Labs, 2024), and encrypted medical diagnostics (Raisaro et al., 2018), and Apple ships FHE-based private contact discovery to 1.5B+ iPhones since iOS 18 (Apple, 2024). For deep-learning workloads, approximate schemes such as CKKS (Cheon et al., 2017) make encrypted inference practical via real-valued arithmetic and SIMD-style slot packing.

Despite this promise, deploying FHE remains bottlenecked by the manual cryptographic-engineering effort each new workload requires[1]: packing, scaling, multiplicative-depth budget, and polynomial-approximation degree are tightly coupled and highly

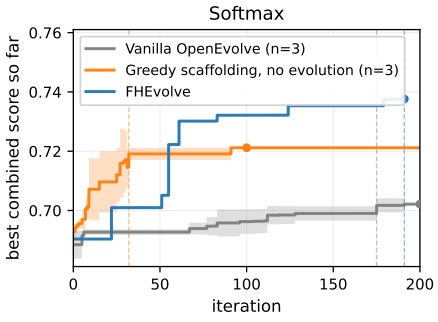

Figure 1: **Generic evolution plateaus; FHEvolve keeps climbing.** Best-so-far score vs. iteration on Softmax from the same seed programs (mean $\pm 1\sigma$; dashed: last gain; greedy runs used a 100-iteration budget; their last value is carried forward).

[1] https://fherma.io/content/68a470c1958b37d5864164eb

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.



(a) Prior code evolution frameworks

(b) Proposed FHEvolve workflow

Figure 2: **Vanilla OpenEvolve (left) vs FHEvolve (right).** Vanilla OpenEvolve requires a hand-written `system_message` per problem and surfaces raw OpenFHE tracebacks unchanged. FHEvolve inserts four components around the same generate–evaluate–select loop: Spec Compiler, Depth-Tiered Strategy Library, Hierarchical Strategy Evolver, and Crypto-Typed Repair Classifier (§4). NEW-badged pieces are introduced by FHEvolve.

non-linear. FHE compilers such as CHET (Dathathri et al., 2019), Orion (Ebel et al., 2025), Fhelipe (Krastev et al., 2024), and EinHops (Garimella et al., 2025) raise the abstraction level, but they lower a program under fixed rules and take the approximation degree, interval, and algorithmic form as human-supplied inputs. Choosing those inputs is the expert search this paper automates; no compiler class searches over which polynomial or decomposition to compute (Cabrero Holgueras, 2023; Krastev et al., 2024).

Evolutionary code-optimization frameworks (Novikov et al., 2025; Sharma, 2025) promise to automate this search, but our experiments show they struggle on FHE workloads. Vanilla OpenEvolve (Figure 2a) requires a hand-written system prompt for every new workload, naming the relevant FHE techniques, depth budget, rotation keys, and target-function design knobs. Even with a tuned prompt, evolution often plateaus early (Figure 1;), and the LM cannot recover when a candidate fails with a typed FHE error (depth exhausted, missing rotation key, accuracy collapse).

The central difficulty is that optimising FHE code requires coordinating approximation accuracy, multiplication depth, data packing, and available rotation keys. A locally promising change can exceed the encryption budget or produce inaccurate outputs. Effective search therefore has to explore different algorithms while respecting these coupled constraints. FHEvolve (Figure 2b) answers each part of this difficulty with one mechanism on the same OpenEvolve loop:

Difficulty	**What FHEvolve contributes**
Cryptographic parameters have consequences a coding model may miss	Extract constraints and compute feasible choices before generation (Spec Compiler, R1)
Different algorithms consume different resources	Filter a reusable strategy library by scheme and depth budget (Strategy Library, R2)
Search repeatedly modifies one approach	Retain candidates from different strategy families (Hierarchical Evolver, R3)
Execution failures give hard-to-interpret feedback	Convert failures into specific repair suggestions (Repair Classifier, R4)

Per workload, FHEvolve takes the specification, the FHE configuration, and seed programs, without a new system prompt for each workload. The seeds and the thirteen strategy templates were written by the authors once and reused. None of the mechanisms adds LM calls, so the call count matches Vanilla and input tokens rise by about 20% (App. L).

Our contributions are:

- **An empirical diagnosis** of LM-driven code generation on FHE workloads (§3): three failure modes (plateaus, per-problem prompt engineering, untyped failure traces) verified across five recent code-generation systems and distilled into requirements R1–R4.
- **The FHEvolve framework** (§4): four mechanisms that change how the search proceeds, not only what the prompt says. They search over the approximation recipe that FHE compilers take as a human-supplied input, and the strategy-family archive dimension applies to any domain whose candidates fall into discrete algorithm families.
- **Results on eight workloads** (§5): seven CKKS FHERMA challenges plus one BFV workload. FHEvolve improves over Vanilla OpenEvolve on Softmax, ReLU, Sign, Eth-

Fraud, and BFV-matvec, ties it on the rest, and scores above FHE-Coder (Kumar et al.) on every workload under our conditions. On Eth-Fraud it gives a fidelity-preserving $1.3\times$ speedup; a $3.6\times$ candidate selected on binary agreement alone does not generalise. Against three FHE compilers, it is the only system that covers all eight workloads without a human-written algorithm.

2 BACKGROUND

2.1 FULLY HOMOMORPHIC ENCRYPTION

FHE provides an evaluation algorithm Eval with $\mathrm{Eval}(f, \mathrm{Enc}(m)) = \mathrm{Enc}(f(m))$, so a server computes on ciphertexts without seeing the plaintext (Gentry, 2009; Cheon et al., 2017). Programs must be static arithmetic circuits of additions and multiplications, since data-dependent control flow is unsupported. Approximate schemes such as CKKS (Cheon et al., 2017) support real-valued arithmetic and pack many values into one ciphertext (SIMD-style slots), which makes them the practical choice for deep-learning workloads. Exact schemes such as BFV/BGV (Brakerski et al., 2014) operate on integers.

Three constraints govern every FHE program and recur throughout this paper. *Multiplicative depth*, the longest multiplication chain, is the dominant cost driver: it bounds the modulus chain and therefore ciphertext size, latency, and key material, and exceeding the depth budget corrupts decryption. Working implementations fix a depth budget up front and optimise around it. *Rotations* shift values across slots and enable reductions and matrix operations, but each rotation index needs a precomputed key, and rotating by an unprovisioned index is a runtime error. *Approximation accuracy*: non-polynomial functions must be replaced by polynomials whose degree consumes depth. Libraries such as OpenFHE (OpenFHE, 2022) expose these primitives, and the developer composes them by hand into a circuit that respects the depth budget, the key set, and a per-slot accuracy tolerance. App. A gives fuller preliminaries.

2.2 EVOLUTIONARY CODE OPTIMISATION AS CONSTRAINED SEARCH

Evolutionary code-optimisation frameworks such as AlphaEvolve (Novikov et al., 2025) and the open-source OpenEvolve (Sharma, 2025) wrap an LM in a generate–evaluate–select loop over a population of candidate programs and improve against a measurable fitness. We build on OpenEvolve and formalise the setting as constrained optimisation. Given an FHE task $\mathcal{T}_{\text{fhe}}$ with a search set $\mathcal{D}_{\text{search}}$, the examples every candidate is scored on during optimisation, the system produces candidate solutions s within a wall-clock budget T and selects under a task-specific metric $M(s; \mathcal{D}_{\text{search}}) \in \mathbb{R}$:

$$s^\star = \arg \max_{\substack{s=\text{Agent}(\mathcal{T},T),\\ s\in\mathcal{S}}} M(s; \mathcal{D}_{\text{search}}), \quad \text{s.t.} \quad \text{gen_time}(s) \le T, \tag{1}$$

where $\mathcal{S}$ is the set of feasible solutions. Because $s^\star$ is selected on $\mathcal{D}_{\text{search}}$, a held-out test set must be disjoint from it. Only Eth-Fraud has one (100 fresh samples, §5.3); the FHERMA challenges ship one fixed test vector each, so the other workloads report search-set scores. The FHERMA challenges we evaluate against use $M = \text{speed} \times \text{accuracy}^2$ with $\text{speed} = T_{\text{ref}}/(T_{\text{ref}} + \text{eval_time})$ and accuracy the per-slot fraction within tolerance. In FHE, feasibility is restrictive: s must respect the depth budget, may only rotate by provisioned indices, and must decrypt within tolerance. The same formalism applies to vanilla OpenEvolve and to FHEvolve. The difference, made precise in §4, is what context the LM receives when proposing the next s.

3 FAILURE MODES OF LM-DRIVEN CODE GENERATION ON FHE WORKLOADS

We document three failure modes that off-the-shelf LM-driven code-generation systems exhibit on FHE workloads. Each is structural: it follows from the absence of any channel through which problem-independent FHE knowledge or typed crypto-failure signals can enter the generate–evaluate–select loop. We surface the modes on FHERMA challenges with OpenEvolve (Sharma, 2025) as the substrate; the same FHE-typed errors recur across five other LM-driven systems (Table 4).

M1: Plateaus across diverse FHE workloads. Evolutionary code search often plateaus early on FHE workloads. On six FHERMA challenges at 190–300 iterations, three show a sharp plateau (GeLU at iteration 61, Softmax at 83, ReLU at 84), while Sign still improves at the end, after which all mutations are local parameter tweaks of whichever strategy family the seed found first. Neither the LM nor the controller has a notion of an FHE strategy family, so nothing forces a structurally different mutation. *Implication: an FHE-aware framework must enforce strategy-family diversity as a structural property of the search* (**R3**).

M2: Per-problem prompt engineering. Each new FHE workload requires a fresh hand-written system prompt naming the techniques the LM should consider. Removing this guidance collapses the run as the LM spends its budget rediscovering basic patterns. Two kinds of problem-independent knowledge must enter every run: the *spec* (scheme parameters and their depth-budget, packing, and rotation-key consequences) and the *strategy catalogue* (standard FHE algorithm patterns with their depth costs). Current systems offer no channel for either. *Implication: both must be derived mechanically from the workload description* (**R1, R2**).

M3: Untyped failures surfaced as raw FHE-library tracebacks. FHE programs fail in a small set of distinctive ways (depth budget exceeded, unprovisioned rotation key, decryption to noise), each with a well-known fix. Off-the-shelf systems forward these as unstructured C++ tracebacks, so the LM that proposed the broken candidate sees no semantic label. Table 4 confirms this across AIDE (Jiang et al., 2025), ML-Master (Liu et al., 2025), RD-Agent (Yang et al., 2025), OpenHands (Wang et al., 2024), and MLAgentBench (Chan et al., 2024): the FHE-specific error rows recur and none exposes them as a typed signal. *Implication: an FHE-aware framework must classify failures into typed buckets and route each to a type-specific repair hint* (**R4**).

Requirements. The three modes share a single root cause and disentangle into four requirements that §4 implements (M2 implies two because its missing knowledge has two forms, spec and strategy):

- **R1 Spec verification (M2).** Derive the workload spec mechanically and catch spec-violators before a $\sim$10 s evaluation.
- **R2 Depth-tier awareness (M2).** Filter strategy proposals against the depth budget so the LM sees a feasible menu.
- **R3 Strategy-space coverage (M1).** Enforce diversity across FHE strategy families structurally.
- **R4 Typed repair (M3).** Classify failures into FHE-typed buckets with type-specific repair hints.

4 THE FHEVOLVE FRAMEWORK

FHEvolve extends OpenEvolve with four components that meet R1–R4 (§3). Each changes what the search does, not only what the prompt says: the Spec Compiler computes feasible choices before generation, the Strategy Library restricts the menu to budget-feasible algorithms, the Hierarchical Evolver keeps one elite per strategy family in the archive, and the Repair Classifier and depth audit turn failures into typed repair signals. None has a counterpart in OpenEvolve or AlphaEvolve. *Inputs.* Per workload, FHEvolve takes the challenge specification, the FHE configuration, and seed programs; the authors wrote three seeds per workload (the same seeds Vanilla uses) and the thirteen strategy templates once, before the runs, and reused them across workloads. The Softmax and BFV seeds already contain a working algorithm; there the search tunes it. None adds LM calls beyond what vanilla OpenEvolve issues. Figure 2b shows how the components sit around the generate–evaluate–select loop. App. N gives the design rationale and full schemas.

FHE specification Compiler (R1). *Design principle: push formal arithmetic out of the LM and into the framework. The compiler chooses the budget, the LM chooses the algorithm.* The split is tractable because the relevant constraints (Paterson–Stockmeyer depth costs, rotation-stride composability) are formal and universal across CKKS workloads. The Spec Compiler is a deterministic, zero-LM preprocessor that reads a challenge directory and emits a structured `Spec` record carrying the literal crypto parameters, workload shape, and a high-level op-type tag on which the Strategy Library is indexed (schema in App. N). A *derived-constraint* pass then emits the largest single-call Chebyshev degree fitting the budget (two-level safety margin) and the shortest two-hop decompositions of

common power-of-two rotation strides into a `Derived FHE constraints` block at the top of the auto-generated prompt. Adding a workload whose op-type already exists costs no curation effort. An unrecognised workload receives the default op-type and the generic activation menu.

Depth-Tiered Strategy Library (R2). *Design principle: do not replace the LM's creativity but prune its option space to a budget-feasible subset before it chooses.* The pruning is tractable because FHE algorithm patterns live on a small Pareto frontier of depth $\times$ rotation count $\times$ accuracy class, and the same pattern instantiates across workloads at the same op-type, so the catalogue is op-type-typed and scheme-typed, each entry's depth cost is known statically, and one `chebyshev_direct` entry serves ReLU, Sigmoid, GeLU, and Sign. Each of thirteen entries (Chebyshev direct fit, $|x|$ decomposition, $\tanh$ decomposition, iterated-squaring $\exp$, log-depth sum reduction, full Softmax pipeline, ...) bundles a parametrised OpenFHE template, a "when to use" heuristic, tunable parameters, and its depth tier, minimum depth budget, and supported schemes. The prompt builder calls `get_strategies` with the spec's scheme and depth budget so only entries satisfying both filters appear in the menu.

Hierarchical Strategy Evolver (R3). *Design principle: match the search hierarchy to the cost-model hierarchy. Outer-level moves pick the depth class, inner-level moves trade off accuracy at fixed depth class.* A strategy swap and a parameter tweak have very different cost profiles in FHE (a swap can change the candidate's depth class), yet vanilla evolution samples both at the same rate; the decomposition is universal across CKKS workloads and recurs in BFV with discrete depth tiers. We realise the hierarchy in two co-designed pieces. First, the auto-generated prompt header lists the strategies returned by the library and instructs the LM to bias mutations toward parameter tuning by default but swap strategy families when the recent fitness trajectory has been flat. Second, the controller-level coupling: a rule-based *strategy classifier* maps every candidate's source to a `strategy_family` index that the evaluator surfaces as a metric, which we add to OpenEvolve's `feature_dimensions` so the MAP-Elites archive maintains one champion *per strategy family* alongside the existing `(eval_time, accuracy)` cells. Promoting strategy choice to a gene of the search preserves diversity even when one family transiently dominates fitness, without changing OpenEvolve's controller code.

Crypto-Typed Repair Classifier (R4). *Design principle: turn opaque domain errors into actionable repair signals on both sides of the evaluation boundary.* Both halves are tractable because FHE failures fall into a small, well-typed set of modes whose regex fingerprints are stable across OpenFHE versions, and the same depth arithmetic the Spec Compiler uses can be turned adversarially to detect doomed candidates. The post-evaluation classifier maps a failed traceback into one of eight typed buckets (e.g. `depth_exhausted`, `rotation_key_missing`), each carrying a one-line repair hint that OpenEvolve's prompt sampler surfaces in the next mutation context. The pre-evaluation *static depth audit* parses the candidate source for declared Chebyshev degrees, iterated-squaring counts, and Newton-iteration counts, sums their depth costs under the same Paterson–Stockmeyer rule, and rejects the candidate before OpenFHE is invoked when the sum exceeds `spec.mult_depth` $- 2$. The audit errs toward letting candidates through: on dynamically computed degrees and loop-based patterns it under-estimates depth, which costs one wasted evaluation. Among the 667 candidates that OpenFHE evaluated successfully, it rejected none; this sample does not establish its reliability beyond them (App. M).

5 Experiments

We answer five questions: (Q1) does FHEvolve overcome the plateau of §3? (Q2) does it apply to each workload without workload-specific prompt editing? (Q3) does each component contribute, and does the evolutionary search contribute beyond them? (Q4) does it run unchanged on a second FHE scheme? (Q5) how does it compare with FHE compilers?

Table 1: Main results across CKKS FHERMA challenges plus one BFV cross-scheme workload. Cell = combined score speed × accuracy^2, higher is better; bold = best per row; mean ± std over n=3 seeds. Last column: eval time of the best Vanilla and FHEvolve programs, seconds per encrypted call (App. C).

*BFV-matvec is an encrypted integer dot product under BFV (16-dim, batched over 16 samples); all other rows are CKKS. †Eth-Fraud Human SoTA is our faithful port of the FHERMA winner, scored under the challenge's binary-agreement metric.

Workload	Human SoTA	Direct Agent	FHE-Coder	Vanilla OE	FHEvolve (ours)	Eval s: Van.→Ours
MatMul‡	0.690 ± 0.005	0.000 ± 0.000	0.580 ± 0.010	0.690 ± 0.003	0.690 ± 0.002	158 → 152
GeLU	0.477 ± 0.005	0.313 ± 0.025	0.510 ± 0.008	0.523 ± 0.002	**0.524** ± 0.002	4.57 → 4.53
Softmax	0.585 ± 0.010	0.374 ± 0.030	0.690 ± 0.012	0.702 ± 0.002	**0.738** ± 0.005	8.48 → **7.11**
ReLU	0.429 ± 0.005	0.873 ± 0.010	0.860 ± 0.008	0.851 ± 0.011	**0.879** ± 0.005	0.87 → **0.57**
Sigmoid	0.679 ± 0.005	0.000 ± 0.000	0.890 ± 0.008	0.895 ± 0.003	**0.895** ± 0.003	0.34 → 0.35
Sign	0.014 ± 0.001	0.123 ± 0.040	0.450 ± 0.015	0.658 ± 0.083	**0.727** ± 0.012	0.97 → **0.77**
Eth-Fraud†	0.696 ± 0.005	0.879 ± 0.008	0.880 ± 0.012	0.891 ± 0.005	**0.968** ± 0.005	0.61 → **0.17**
BFV-matvec*	0.988 ± 0.003	0.004 ± 0.000	0.920 ± 0.012	0.985 ± 0.005	**0.991** ± 0.003	0.030 → **0.018**

5.1 Workloads and baselines

We evaluate on eight workloads. Seven are CKKS FHERMA challenges[2] that the privacy-preserving-ML community treats as canonical bottlenecks of encrypted inference: linear algebra (MATMUL), polynomial activations (GELU, SOFTMAX, RELU, SIGMOID), a comparison primitive (SIGN), and an end-to-end binary classification task, encrypted linear-SVM inference on the FHERMA Ethereum fraud dataset (ETH-FRAUD; §5.3). The eighth (BFV-MATVEC; §5.3) is a 16-dimensional encrypted integer dot product under BFV, a demonstration on one additional scheme and workload. Each FHERMA challenge ships with a reference, a public accuracy threshold, and a human leaderboard (App. D).

We compare against four baselines. (i) HUMAN SOTA, the top human FHERMA leaderboard solution for each challenge, re-run by us under our evaluation conditions (for Eth-Fraud, our port of the winning solution). These cells are reference points under our conditions, not new leaderboard entries. (ii) DIRECT AGENT, a single-shot baseline that prompts `gpt-5` once with the extracted spec and the same FHE op-list and cost model FHEvolve uses, without the strategy library or hierarchical directive, at K=5 best-of-K. Vanilla and FHEvolve differ from it in both search budget (200 iterations vs. 5 candidates) and guidance, so a gap over it is not attributable to either alone. (iii) VANILLA OE, OpenEvolve (Sharma, 2025) with hand-written per-problem prompts and three diverse seed programs per workload. (iv) FHE-CODER (Kumar et al.), the strongest prior LM-driven FHE code-generation system, an agentic generator with iterative compilation, validation, and feedback, run as released with its default model and feedback budget. It uses no population search. §5.4 adds three FHE compilers, Orion, HEIR, and CHEHAB, on all eight workloads.

5.2 Experimental settings

FHEvolve and Vanilla share OpenEvolve as the engine, with FHEvolve adding the four components of §4. Both use three islands, a per-island MAP-Elites archive of size 30, SEARCH/REPLACE diff mutation, and 200 iterations. Mutation prompts call `openai/gpt-5` via OpenRouter (temperature 0.7, two instances weighted 0.8/0.2). FHEvolve makes the same number of LM calls as Vanilla, with about 20% more input tokens (App. L). Each candidate is evaluated by a real OpenFHE-Python run with parameters from the FHERMA `config.json` under the metric of §2.2. Main-table cells report mean±std over n=3 seeds. Ablation cells (§5.5) are runs from one batch.

5.3 Main results

Table 1 summarises combined scores across the eight workloads. Three patterns are visible. *First, FHEvolve strictly improves over Vanilla on the high-headroom workloads*: $+5.0\%$ on Softmax (**0.738** vs. 0.702 ± 0.002), where it climbs past Vanilla's plateau and is still improving at iteration 191 with 16% faster eval time; $+3.3\%$ on ReLU (**0.879** vs. 0.851 ± 0.011) with 34% faster eval; and

[2] `https://fherma.io/challenges`

$+10\%$ on Sign (**0.727** vs. 0.658 ± 0.083) with a seven-fold smaller seed variance. *Second, on the depth-saturated workloads (Sigmoid, depth 7; GeLU, depth 29 with a polynomial fit that saturates well below speed=1) FHEvolve ties Vanilla within noise* (Sigmoid 0.895 vs. 0.895 ± 0.003; GeLU 0.524 vs. 0.523 ± 0.002). *Third*, iterated search is far above the five-candidate Direct Agent on Softmax, Sigmoid, and GeLU (0.738, 0.895, 0.524 vs. 0.374, 0.000, 0.313), a gap from both budget and guidance, but not on ReLU, where Direct Agent reaches 0.873 (App. C).

End-to-end binary classification. Eth-Fraud is an end-to-end encrypted ML pipeline: a linear SVM trained offline on 7,856 transactions; 16 balanced transactions packed into one CKKS ciphertext form the search set, and 100 further transactions, never used for selection, form the held-out test set. The seed performs a batched encrypted matrix-vector product and a degree-31 Chebyshev sigmoid, and already saturates accuracy. Two selection criteria give two results:

Selection criterion	Eval time	What the result establishes
Agreement + max prob. error < 0.01	$0.61 \rightarrow 0.47$ s ($1.3\times$)	fidelity preserved; 100/100 held-out
Binary agreement only (challenge)	$0.61 \rightarrow 0.168$ s ($3.6\times$)	probabilities change; 94/100 held-out

The $1.3\times$ candidate comes from re-ranking the candidates already explored under the challenge metric by the stricter criterion, not from a separate run. About half of the $3.6\times$ is standard FHE micro-optimisation; the other half is a *metric-aware shortcut*, a degree-3 sigmoid that keeps every search-set decision but raises the mean probability error from 6×10^{-4} to 0.22. Improving the search score can fail to preserve the intended output, so the evaluator must encode it (App. E).

A second scheme: BFV. BFV-MATVEC re-uses the FHEvolve pipeline unchanged: the Spec Compiler reads BFV parameters, the library filter passes BFV entries, and the Repair Classifier types BFV failures. BFV arithmetic is exact, so accuracy is binary and the headroom is in eval time. FHEvolve reaches **0.991** (0.018 s, 100% exact match) against the best seed's 0.985. On this one BFV workload the components ran without redesign; broader cross-scheme generalisation is untested (App. F).

5.4 COMPARISON WITH FHE COMPILERS

We ran three FHE compilers on all eight workloads: Orion (Ebel et al., 2025), a CKKS deep-learning compiler on Lattigo; HEIR (Ali et al., 2025), an MLIR compiler for CKKS and BFV on OpenFHE; and CHEHAB (Seddiki et al., 2026), a BFV compiler with an e-graph vectoriser on SEAL. We gave each compiler its best hand-tuned configuration (App. G). Depth, the number of levels consumed, is a property of the algorithm rather than the library.

Table 2: FHEvolve vs. three FHE compilers on all eight workloads. Cell = accuracy at the challenge tolerance / multiplicative depth, within the challenge depth budget (in brackets). ^{h}the compiler cannot build the circuit; a human wrote the algorithm on its low-level API. ^{o}best within budget; the compiler reaches higher accuracy only above the budget. ^{w}needs a human workaround for a compiler bug. ✗ = not expressible. Every compiler cell also needed a human-chosen degree and interval; FHEvolve needed none.

Workload [budget]	Orion	HEIR	CHEHAB	FHEvolve
GeLU [29]	100 / 6	100 / 8^h	✗	100 / 6
ReLU [12]	99.6 / 10	99.3 / 9	✗	99.0 / 9
Softmax [29]	100 / 10^h	100 / 21^h	✗	100 / 21
Sigmoid [7]	100 / 7	49.1 / 7°	✗	100 / 7
Sign [10]	99.2 / 10	8.0 / 8°	✗	95.5 / 10
MatMul [29]	100 / 2^h	100 / 2^h	100 / 2^h	100 / 2
Eth-Fraud [10]	100 / 4	100 / 6^w	✗	100 / 5
BFV-matvec [4]	✗	100 / 1	100 / 1	100 / 1

Table 2 supports three findings. *Coverage.* FHEvolve is the only system that produces a within-budget program for all eight workloads without a human-written algorithm. No compiler builds Softmax or

ciphertext MatMul: Orion and HEIR lack the operations, CHEHAB's vectoriser exhausts 18 GB, and all three ran only on kernels we wrote by hand. *Quality where a compiler has a recipe.* FHEvolve matches the best compiler accuracy on GeLU, Sigmoid, Eth-Fraud, and BFV-matvec at equal or lower depth, except on Eth-Fraud, where Orion uses one level fewer. It trails on two workloads. On ReLU, Orion and HEIR are 0.6 and 0.3 points more accurate. On Sign, a hand-tuned Orion reaches 99.2% against our 95.5% at the same depth, after we swept 35 configurations. Shipped defaults reach 0–63% on the CKKS activations; the gap to the tuned cells is the human effort FHEvolve replaces. *Latency.* HEIR shares our OpenFHE backend, so at a matched ring dimension its latency is comparable. FHEvolve is 1.5–$2.5\times$ faster on GeLU, ReLU, Softmax, and Eth-Fraud. It is 1.3–$1.6\times$ slower on BFV-matvec and MatMul, where both run the same algorithm and HEIR emits C++ rather than Python (App. G). Compilers optimise a given recipe well; FHEvolve supplies the recipe.

5.5 Ablations: components and the evolutionary mechanism

Table 3 reports two complementary ablations on the two headroom workloads (all six in App. I). The *component* ablations disable one component at a time under an otherwise identical configuration: −Spec reverts to the hand-written prompt; −Strategy deletes the menu; −Hier drops the swap-then-tune directive and the strategy-family archive dimension; −Repair surfaces raw tracebacks and disables the depth audit. The *non-evolutionary* ablation keeps all four components and removes the population (one island, archive of size one, every parent the current best, no population members in the prompt), so the search is greedy refinement with the same LM, seeds, evaluator, and budget.

Table 3: Ablations on the two headroom workloads at a matched budget of 90 iterations: combined score, eval time of the best program (seconds per encrypted call), and its slot accuracy. Every arm starts from the same three seed programs. "Full (190)" is the full system at its complete budget. Bold marks the best score and the lowest eval time among the 90-iteration arms. Four further workloads, whose seeds sit at the depth ceiling, are in App. I.

	Softmax			ReLU		
Variant (90 iters)	**Score**	**Eval (s)**	**Acc.**	**Score**	**Eval (s)**	**Acc.**
FHEvolve (full)	**0.732**	**6.89**	99.2%	0.863	0.71	99.2%
−Spec Compiler	0.697	8.70	100.0%	0.866	0.69	99.2%
−Strategy Library	0.694	8.80	100.0%	0.857	0.77	99.5%
−Hierarchical Evolver	0.697	8.70	100.0%	**0.877**	**0.59**	99.0%
−Repair Classifier	0.698	8.65	100.0%	0.826	0.90	98.7%
Non-evo (greedy)	0.722	7.43	99.5%	0.800	0.80	96.4%
Vanilla OE	0.702	8.49	100.0%	0.846	0.87	99.6%
FHEvolve (full, 190)	0.738	7.12	100.0%	0.879	0.57	99.0%

Components. At a matched budget the full system is the best Softmax arm on score and the fastest at the accuracy threshold: 19–22% faster than every single-component ablation and 19% faster than Vanilla. On ReLU removing the Repair Classifier gives the largest observed drop: the score from 0.863 to 0.826 and slows the best program from 0.71 to 0.90 s, since ReLU raises the FHE-typed errors it repairs; on Softmax its removal costs 0.034. The other three ReLU removals fall within 0.014 of Full on either side. The Spec Compiler's ablation substitutes a hand-written prompt, so parity is its intended outcome. On the four workloads in App. I the seeds sit at the depth ceiling, no arm moves more than 0.01, and several ablations score marginally above Full: the components have no measurable effect there.

Evolutionary mechanism. The non-evolutionary ablation isolates the population search from the scaffolding. At the matched budget, greedy refinement with all four components reaches 0.722 on Softmax and never improves on ReLU (0.800, the seed score), while the full system reaches 0.732 and 0.863. Over the shared seeds (Figure 1), Vanilla gains 0.013 on Softmax and 0.040 on ReLU, greedy scaffolding 0.028 and 0.000, and FHEvolve 0.047 and 0.077 at full budget; the greedy variant plateaus by iteration 32, FHEvolve is still improving at 191 and 187. On the ceiling workloads the evidence is weaker: greedy refinement trails Full by 0.009 on Eth-Fraud and leads it by 0.005 on GeLU (App. I).

Headroom. The gain over Vanilla tracks search headroom, not op-type: Eth-Fraud ($+0.077$), Sign ($+0.069$), Softmax ($+0.036$), and ReLU ($+0.028$) admit many strategies; the other four hit their depth ceiling at once.

Correctness scope. FHEvolve is heuristic search with empirical validation, not verified synthesis. For an approximate CKKS program the certifiable object is a sup-norm error bound over the input range, not equivalence to an input program. We certify it by a Lipschitz bound on a 2×10^7-point grid: GeLU $< 8.18\times10^{-4}$ on $[-6.79, 7.06]$, Sigmoid $< 9.40\times10^{-4}$ on $[-25, 25]$, and ReLU $< 2.6\times10^{-3}$ by composition. Bounds hold on the fitted range only (App. J).

6 Related Work

FHE compilers. FHE compilers fall into five classes. Rule-based systems and IR frameworks (EVA, CHET (Dathathri et al., 2019), HECO (Viand et al., 2023), Hecate (Lee et al., 2022), HEIR (Ali et al., 2025)) apply designer-written rewrite and scale-management rules. Analytical and packing compilers (Fhelipe (Krastev et al., 2024), Orion (Ebel et al., 2025), EinHops (Garimella et al., 2025)) choose tensor layouts and rotation schedules from cost models. Synthesis and search-based compilers (Porcupine (Cowan et al., 2021), Coyote (Malik et al., 2023)) search over BFV vectorisation schedules. Multi-engine and learned systems (CHEHAB (Seddiki et al., 2026), CHEHAB-RL (Sefsaf et al., 2026)) select algebraic rewrites by e-graphs or a trained policy. All five classes share one property: the function the program computes is fixed before optimisation, and the polynomial degree, fit interval, and algorithmic decomposition of every non-polynomial operation are inputs a human supplies. FHEvolve searches over the approximation recipe itself, which is why every compiler we ran needed a human-chosen degree and interval on each CKKS workload (App. H), and why its correctness object is an error bound rather than an equivalence proof (§5.5). §5.4 compares against three of them, Orion, HEIR, and CHEHAB, on all eight workloads.

Agentic and evolutionary code generation. Single-shot generation (Chen et al., 2021; Hendrycks et al., 2021; Austin et al., 2021) produces candidates from one prompt (our Direct Agent baseline). Agentic scaffolds (AIDE (Jiang et al., 2025), ML-Master (Liu et al., 2025), RD-Agent (Yang et al., 2025), OpenHands (Wang et al., 2024), MLAgentBench (Chan et al., 2024)) execute, observe, and revise with raw tracebacks; evolutionary frameworks (Novikov et al., 2025; Sharma, 2025) add a population. All are problem-agnostic (Table 4).

LM-assisted FHE code generation. FHE-Coder (Kumar et al.) is the closest baseline: like FHEvolve it targets FHERMA challenges, and it refines a candidate through iterative compilation, validation, and feedback. It keeps no population, so it has no strategy-family diversity and no depth-aware pre-filter. Under our evaluation conditions (Table 1), FHEvolve scores higher than FHE-Coder on every workload.

7 Conclusion

Vanilla evolutionary code search struggles on FHE workloads because no channel carries FHE constraints or typed failure signals into the generate–evaluate–select loop. **FHEvolve** adds four mechanisms that compute feasible choices before generation, restrict the strategy menu, keep one elite per strategy family, and type failures into repair hints. It improves over Vanilla where there is headroom, ties it elsewhere, and scores above FHE-Coder on every workload under our conditions; on Eth-Fraud its fidelity-preserving speedup is $1.3\times$. Against three FHE compilers it alone covers all eight workloads without a human-written algorithm, though a hand-tuned compiler is more accurate on Sign and ReLU. The strategy-family archive dimension and the typed failure channel apply beyond FHE. Limitations: the author-written seeds and thirteen-entry library bound FHEvolve's autonomy, ablations are single-seed, only Eth-Fraud has a held-out test set, bounds hold on the observed range, and bootstrapped CKKS is out of scope.

AI USE STATEMENT

LLMs were used for improving grammar, ideation/brainstorming and implementation of the code. All LLM generated code was reviewed thoroughly by the authors. All experimental decisions, claims, and final text are the authors' responsibility

ETHICS STATEMENT

No ethical concerns are raised by this work.

REPRODUCIBILITY STATEMENT

To ensure reproducibility we have used seeded runs and we will open-source the code upon acceptance.

REFERENCES

Asra Ali, Jaeho Choi, Bryant Gipson, Shruthi Gorantala, Jeremy Kun, Wouter Legiest, Lawrence Lim, Alexander Viand, Meron Zerihun Demissie, and Hongren Zheng. Heir: A universal compiler for homomorphic encryption. *arXiv preprint arXiv:2508.11095*, 2025.

Apple. Homomorphic encryption. `https://machinelearning.apple.com/research/homomorphic-encryption`, 2024. Accessed: 2025.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory*, 6(3):1–36, 2014.

José Cabrero Holgueras. Towards improved homomorphic encryption for privacy-preserving deep learning. 2023.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*, pp. 409–437. Springer, 2017.

Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

Meghan Cowan, Deeksha Dangwal, Armin Alaghi, Caroline Trippel, Vincent T Lee, and Brandon Reagen. Porcupine: A synthesizing compiler for vectorized homomorphic encryption. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 375–389, 2021.

Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. Chet: an optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, pp. 142–156, 2019.

Austin Ebel, Karthik Garimella, and Brandon Reagen. Orion: A fully homomorphic encryption framework for deep learning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 734–749, 2025.

Karthik Garimella, Austin Ebel, and Brandon Reagen. Einhops: Einsum notation for expressive homomorphic operations on rns-ckks tensors. In *Proceedings of the 13th Workshop on Encrypted Computing & Applied Homomorphic Computing*, pp. 50–61, 2025.

Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pp. 201–210. PMLR, 2016.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N Wright. Privacy-preserving machine learning as a service. *Proceedings on Privacy Enhancing Technologies*, 2018.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Aleksandar Krastev, Nikola Samardzic, Simon Langowski, Srinivas Devadas, and Daniel Sanchez. A tensor compiler with automatic data packing for simple and efficient fully homomorphic encryption. *Proceedings of the ACM on Programming Languages*, 8(PLDI):126–150, 2024.

Mayank Kumar, Jiaqi Xue, Mengxin Zheng, and Qian Lou. Fhe-coder: Benchmarking secure agentic code generation for fully homomorphic encryption. In *The Fourteenth International Conference on Learning Representations*.

Yongwoo Lee, Seonyeong Heo, Seonyoung Cheon, Shinnung Jeong, Changsu Kim, Eunkyung Kim, Dongyoon Lee, and Hanjun Kim. Hecate: Performance-aware scale optimization for homomorphic encryption compiler. In *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 193–204. IEEE, 2022.

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Weinan E, and Siheng Chen. Ml-master: Towards ai-for-ai via integration of exploration and reasoning, 2025. URL `https://arxiv.org/abs/2506.16499`.

Varun Madathil and Alessandra Scafuro. Prifhete: Achieving full-privacy in account-based cryptocurrencies is possible. *Cryptology ePrint Archive*, 2023.

Raghav Malik, Kabir Sheth, and Milind Kulkarni. Coyote: A compiler for vectorizing encrypted arithmetic circuits. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 118–133, 2023.

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.

OpenFHE. Openfhe: Open-source fully homomorphic encryption library. `https://openfhe.org/`, 2022. Accessed: 2026-03-12.

Jean Louis Raisaro, Gwangbae Choi, Sylvain Pradervand, Raphael Colsenet, Nathalie Jacquemont, Nicolas Rosat, Vincent Mooser, and Jean-Pierre Hubaux. Protecting privacy and security of genomic data in i2b2 with homomorphic encryption and differential privacy. *IEEE/ACM transactions on computational biology and bioinformatics*, 15(5):1413–1426, 2018.

Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

Abdessamed Seddiki, Arab Mohammed, Zakaria Hebbal, Aimad Chabounia, Eduardo Chielle, Karima Benatchba, Challal Yacine, Djamel Eddine Menacer, Michail Maniatakos, and Riyadh Baghdadi. Chehab: Automatic compiler code optimization for fully homomorphic encryption. In *Proceedings of the 35th ACM SIGPLAN International Conference on Compiler Construction*, pp. 81–93, 2026.

Bilel Sefsaf, Abderraouf Dandani, Abdessamed Seddiki, Arab Mohammed, Eduardo Chielle, Michail Maniatakos, and Riyadh Baghdadi. Chehab rl: Learning to optimize fully homomorphic encryption computations. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 342–360, 2026.

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025. URL `https://github.com/algorithmicsuperintelligence/openevolve`.

Alexander Viand, Patrick Jattke, Miro Haller, and Anwar Hithnawi. {HECO}: Fully homomorphic encryption compiler. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 4715–4732, 2023.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

Xu Yang, Xiao Yang, Shikai Fang, Bowen Xian, Yuante Li, Jian Wang, Minrui Xu, Haoran Pan, Xinpeng Hong, Weiqing Liu, et al. R&d-agent: Automating data-driven ai solution building through llm-powered automated research, development, and evolution. *arXiv e-prints*, pp. arXiv–2505, 2025.

Z1Labs. What is fhevm? `https://docs.z1labs.ai/cyphers-fhevm-technology/what-is-fhevm`, 2024. Accessed: 2025.

A FHE Preliminaries

Industry-relevant FHE schemes split into boolean-gate systems such as TFHE (Chillotti et al., 2020) and arithmetic schemes such as BGV/BFV (Brakerski et al., 2014) and CKKS (Cheon et al., 2017). Because the plaintext is obscured, data-dependent control flow (branching, plaintext-indexed loops) is unsupported; programs must be unrolled into static arithmetic circuits composed of additions and multiplications. CKKS extends this to real-valued arithmetic and packs many values into a single ciphertext (SIMD-style slots).

Multiplicative depth. All efficient FHE schemes are built on noise-based hardness assumptions (Regev, 2009), and each homomorphic operation consumes noise budget; once the budget is exceeded, decryption returns garbage. Multiplicative depth, the length of the longest multiplication chain, bounds the modulus chain size, which in turn bounds ciphertext size, evaluation latency, and the size of the keying material. Bootstrapping can refresh the noise budget mid-circuit but is expensive in practice, so working FHE implementations operate within a fixed depth budget chosen up front and optimise everything else around it.

Rotations. Packed schemes expose cyclic rotations ($\mathrm{EvalRotate}$) that shift values across slots, enabling sum reductions, broadcasts, and diagonal-style matrix operations. Each rotation index requires a precomputed evaluation key, and calling $\mathrm{EvalRotate}$ with an index outside the provisioned key set is a runtime error.

Polynomial approximation. Non-polynomial functions (ReLU, GeLU, sigmoid, sign, reciprocal) must be approximated by polynomials. Under Paterson–Stockmeyer evaluation a degree-d Chebyshev series costs about $\lceil \log_2(d+1) \rceil$ multiplicative levels, so degree trades directly against depth. Production implementations use libraries such as OpenFHE (OpenFHE, 2022), where the developer composes additions, multiplications, rotations, and polynomial evaluations into a circuit that respects the depth budget, the provisioned rotation keys, and a per-slot accuracy tolerance.

B Cross-System Failure-Mode Catalogue

Table 4 catalogues which FHE-typed errors recur across five recent LM-driven code-generation systems applied to our FHERMA workloads. The three FHE-specific rows recur across systems but none of them exposes the failure as a typed signal the LM can act on. Generic code-generation errors (wrong API names, undeclared variables) appear in all five systems and are omitted.

Table 4: FHE-typed failure categories observed across five recent LM-driven code-generation systems applied to our FHERMA workloads. Without typed repair, every row collapses into "the candidate failed."

Error Category	AIDE	ML-Master	RD-Agent	OpenHands	MLAgentBench
Methods on Ciphertext object	✓✓	×	×	×	✓
Multiplicative depth exceeded	×	✓	✓	×	×
Missing rotation keys	×	✓	✓	✓	×

✓✓= dominant failure mode, ✓= observed, × = not observed

C Per-metric Breakdown of Main Results

The combined score collapses two axes (eval-time speed and per-slot accuracy) into one number, score = speed × accuracy^2 where speed = $T_{\text{ref}}/(T_{\text{ref}} + t)$. The reference latencies T_{ref} are workload-specific (Matmul 60s, Softmax 20s, GeLU 5s, ReLU 5s, Sigmoid 3s, Sign 3s; chosen during the evaluator smoke-tests so that the reference solution scores ~ 0.5). Tables 5 and 6 decompose the headline numbers so the source of every win is visible.

Table 5: Accuracy breakdown per workload. Each cell is the fraction of output slots within the FHERMA tolerance (10^{-3} per slot for vector outputs, binary for scalar outputs). Dashes mark cells where the per-metric log was not retained; the combined score in Table 1 is unaffected.

Workload	Human SoTA	Direct Agent	FHE-Coder	Vanilla OE	FHEvolve (ours)
Matrix Multiplication	100.0%	–	100.0%	100.0%	100.0%
GeLU	100.0%	100.0%	100.0%	100.0%	100.0%
Softmax	100.0%	84.4%	100.0%	100.0%	100.0%
ReLU	100.0%	99.3%	100.0%	99.6%	99.0%
Sigmoid	87.5%	0.0%	100.0%	100.0%	100.0%
Sign	16.3%	48.7%	–	92.9% ± 2.8%	–

Table 6: Eval-time breakdown per workload (seconds per single encrypted-input call, measured on a single CPU thread inside the `openfhe-python` evaluator). Lower is better. Sigmoid Direct-Agent eval-time is ∞ (every K=5 candidate failed the depth budget before returning).

Workload	Human SoTA	Direct Agent	FHE-Coder	Vanilla OE	FHEvolve (ours)
Matrix Multiplication	27.0 s	–	43.4 s	27.0 s	27.0 s
GeLU	5.48 s	10.97 s	4.80 s	4.57 s	4.53 s
Softmax	14.17 s	18.06 s	9.0 s	8.48 s	7.11 s
ReLU	6.66 s	0.65 s	0.85 s	0.87 s	0.57 s
Sigmoid	0.38 s	∞	0.35 s	0.34 s	0.35 s
Sign	2.74 s	2.78 s	–	0.97 ± 0.49 s	–

The decomposition shows two distinct mechanisms behind FHEvolve's wins. *Speed-driven gains.* On Softmax, FHEvolve and Vanilla OE both reach 100% accuracy, but FHEvolve evaluates in 7.11 s vs. 8.48 s, a 16% speedup that, at $T_{\text{ref}} = 20$ s, produces the entire $+5.0\%$ combined-score gap. The same shape appears on ReLU (0.57 s vs. 0.87 s). *Accuracy-driven gains.* On Sigmoid, FHE-Coder and FHEvolve both maintain 100% accuracy, but the Human SoTA Chebyshev fit at degree 63 only reaches 87.5% under our depth-7 test case: the Human SoTA is depth-bottlenecked by accuracy, not speed.

D Workload-Selection Rationale

Why these primitives. The selection reflects the dominant cost contributors of encrypted ML inference. *Linear algebra* (matrix multiplication) is the workhorse of every dense and convolutional layer. *Polynomial activations* are the single hardest piece of CKKS-based ML inference: every

non-polynomial function must be approximated by a low-depth polynomial, and the depth/accuracy trade-off dominates achievable latency. *Comparison primitives* (sign) are the building block for argmax, max-pooling, attention masking, and the $|x|$-decomposition of ReLU. Three of the six per-primitive challenges are FHERMA *white-box* (matmul, GeLU, Softmax) and three are *black-box* (ReLU, Sigmoid, Sign).

Why FHERMA. The FHERMA challenge series posts each primitive as a standalone problem with a reference implementation, a public correctness requirement, and a human leaderboard. The homomorphic-encryption-standardization benchmark initiative[3] has similarly converged on end-to-end encrypted ML inference as a flagship workload, and the linear-algebra plus polynomial-approximation primitives we evaluate are its internal building blocks.

E ETH-FRAUD OPTIMISATION BREAKDOWN

The seed `solve` performs (i) a batched encrypted matrix-vector product (Hadamard with a tiled weight plaintext, then a log-depth stride-64 rotate-and-sum reduction) and (ii) a degree-31 Chebyshev sigmoid over $[-15, 15]$. The evolved program decomposes into two effects.

Standard FHE micro-optimisations (roughly half the gain). The rotate-and-sum tree is pruned from six rotations to four: only the bit-set positions of `N_FEATURES`=45 contribute ($45 = 32 + 8 + 4 + 1$), so rotations by 16 and 2 are skipped. The weight plaintext is cached on the function object instead of rebuilt every call. The bias is added as a scalar broadcast rather than a packed-plaintext tile.

Metric-aware shortcut (the remaining half). The LM replaces the degree-31 sigmoid over $[-15, 15]$ with a degree-3 fit over $[-2.087, -1.411]$. The replacement preserves `sign(prob - 0.5)` on every test sample but not the probability itself: mean per-slot $|\text{prob}_{\text{FHE}} - \text{prob}_{\text{plaintext}}|$ rises from 6×10^{-4} (seed) to 0.22 (evolved). The optimiser is maximising the metric it is given, and the challenge's binary-classification metric admits this shortcut.

Fidelity-bounded re-scoring. The evaluator logs the maximum per-slot probability error per candidate, so the 139 explored candidates can be re-ranked under the stricter constraint $\max |\text{prob}_{\text{FHE}} - \text{prob}_{\text{plaintext}}| < 0.01$ without re-running OpenFHE (Table 7). This is a lower bound on what a run that uses the fidelity-bounded metric as its fitness could find.

Table 7: Eth-Fraud candidates under the challenge metric and under a fidelity-bounded metric. Held-out check: 100 fresh samples, 50 per class, disjoint from the 16 the search scored against.

Selection	Eval time	Agreement (16)	Max prob. error	Agreement (100 held-out)
Seed (degree-31 sigmoid)	0.61 s	100%	6×10^{-4}	100%
Best under binary agreement	0.17 s	100%	0.30	94%
Best with max prob. error < 0.01	0.47 s	100%	6×10^{-4}	100%

The shortcut solution does not generalise: on the 100 held-out samples it agrees on 94% and its probability error grows to about 15, because it was fit to the narrow score range spanned by the 16 samples. The fidelity-preserving solution matches the plaintext SVM on all 100 with maximum probability error 6×10^{-4}. We measure agreement against the plaintext SVM rather than ground truth, since the SVM's own classification accuracy is a property of the challenge's model, not of the FHE optimisation.

F BFV-MATVEC DETAILS

Sixteen integer test inputs are batched at stride 16 into a single BFV ciphertext, and the evolved `solve` returns the integer scalar $y_i = \sum_j W[j]\, x_i[j]$ at slot $i \cdot 16$ for each sample. FHEvolve reaches combined score 0.991 (eval_time 0.018 s, 100% exact match) at iteration 124, against the

[3] `https://fhe-benchmarking.github.io/`

best seed's 0.985, by shaving ∼40% off eval time. The Direct-Agent baseline collapses to 0.004 (6.25% accuracy): the single-shot LM produces a syntactically valid `solve` but returns the right answer only at slot 0 and zeros at slots 16, 32, . . ., 240.

G Compiler Comparison Details

Protocol. Every compiler ran on the exact challenge test vectors, and we computed accuracy after decryption with our evaluator's metric. We gave each compiler its best configuration: we swept degrees and schedules within the challenge depth budget and report the best cell, alongside the shipped default. We record every human input a compiler needed. Depth is the number of levels the circuit consumes. For Orion we measure the level drop of the ciphertext, with no bootstrap placed. For HEIR we read the depth it chose from its generated OpenFHE code. For FHEvolve we measure the output level, which equals the smallest depth budget under which the program decrypts correctly. All scripts and logs are in the released code.

Orion (Lattigo backend; the OpenFHE and HEAAN backends are stubs). *Fitting bug.* Orion's `fit()` calls NumPy's Chebyshev fit without its domain argument, so its coefficients are fitted on a slightly shrunken interval. This gives 62.6% on GeLU at degree 24, and 100% once the domain is corrected; Table 2 uses the corrected fit. *ReLU.* Orion computes $x\,(\mathrm{sign}(x)+1)/2$ with a composite minimax sign. Schedule $[15, 27]$ gives 99.6% at depth 10; the default $[15, 15, 27]$ needs depth 14, over the budget of 12. FHEvolve's $\frac{1}{2}(x+\sqrt{x^2})$ runs at depth 9. The $\sqrt{x^2}$ identity is known; Orion does not ship it. *Sigmoid.* Degree 51 is the smallest to reach 100% at depth 7; degree 50 gives 93% and the shipped degree 31 gives 12.8%. *Sign.* We swept 35 schedule and dead-zone settings; $[31, 31]$ reaches 99.2% at depth 10, at the ceiling set by inputs near zero. Standalone use of Orion's sign module overflows, so we added a two-line wrapper. *Eth-Fraud.* A native linear layer plus sigmoid of degree 5 gives 100% class agreement at depth 4. *Softmax and MatMul.* Orion has no exp, reciprocal, or division, and its linear layers take plaintext weights only. We composed Softmax by hand (exp of degree 31, rotate-and-sum, reciprocal of degree 7 on a human-chosen interval) and wrote the MatMul kernel by hand (896 rotations); neither is a compiler result. *BFV.* Orion supports CKKS only.

HEIR (pip release 2026.9.1, OpenFHE backend). *Native.* ReLU (built-in $\max(x, 0)$ fit, degree 127), Sigmoid, Eth-Fraud, and BFV-matvec compile from high-level ops. Within the Sigmoid budget of 7 the best fit reaches 49.1%; 100% needs depth 8. Eth-Fraud reduces 45 features incorrectly (81% class agreement) unless a human pads them to 64. *Human algorithm.* HEIR has no GeLU op; with a human-written erf formula its built-in erf fit reaches 100% at depth 8. Its native sign fit tops out at 8.0% within budget; a human surrogate with sharpening reaches 98.2% only at depth 17. Every native Softmax form and `linalg.matmul` at every size fail inside the compiler; we wrote both by hand. Above ten levels HEIR inserts bootstraps that crash in OpenFHE, so every deep configuration sets `greedy-level-budget=29`. HEIR's defaults (degree 5 on $[-1, 1]$) score 0–7% on the CKKS activations.

CHEHAB (BFV only, SEAL backend). On BFV-matvec, the e-graph vectoriser turns a scalar program into a correct circuit (16/16 exact, depth 1) in 0.13–0.17 s; a human-vectorised program runs in 0.018 s, level with FHEvolve. On MatMul, the vectoriser exhausts 18 GB and crashes at 32×32; a hand-vectorised kernel (4,096 ciphertext products, 36,864 rotations) is exact after quantisation but takes 1,949 s. CHEHAB's data type is an integer vector, so the six CKKS workloads are not expressible.

Latency. Table 8 compares HEIR and FHEvolve at a matched ring dimension on the same OpenFHE library. HEIR does not let the user set the ring, so we ran FHEvolve's programs at HEIR's ring and modulus sizes. Orion (Lattigo) and CHEHAB (SEAL) use other libraries and are not comparable.

H Applicability of FHE Compilers to the Workloads

Table 9 lists, for each compiler named in the Related Work, which of our eight workloads it accepts. The first three rows are measured (§5.4, App. G); the rest are from each system's documentation

Table 8: HEIR vs. FHEvolve latency at a matched ring dimension N, both on OpenFHE, seconds per encrypted call, median of 3. Sigmoid and Sign are omitted because HEIR has no within-budget program that reaches the tolerance.

Workload	N	HEIR	FHEvolve	Same algorithm?
GeLU	2^{15}	0.44	**0.28**	no (erf form, depth 8 vs. 6)
ReLU	2^{15}	1.47	**0.59**	no (max fit vs. $\sqrt{x^2}$)
Softmax	2^{16}	3.06	**2.07**	no (both human-designed pipelines)
Eth-Fraud	2^{15}	0.35	**0.20**	no (degree 9 vs. 3 sigmoid)
MatMul	2^{14}	**2.16**	3.52	yes (BSGS, 32 ciphertext products)
BFV-matvec	2^{13}	**0.009**	0.012	yes (1 product, 4 rotations)

and input language. Every system takes the function and its approximation recipe as given: where a compiler accepts a CKKS activation, a human chooses its degree and interval.

Table 9: Applicability of FHE compilers to our workloads.

System	Class	Scheme / input	Applicable workloads and reason
Orion (run)	analytical, ML	CKKS, PyTorch modules	GeLU, ReLU, Sigmoid, Sign, Eth-Fraud natively; Softmax, MatMul only as hand-written kernels; no BFV
HEIR (run)	IR framework	CKKS/BFV, MLIR	ReLU, Sigmoid, Eth-Fraud, BFV-matvec natively; GeLU, Softmax, Sign, MatMul need human-written algorithms
CHEHAB (run)	multi-engine	BFV, C++ DSL	BFV-matvec natively; MatMul only as a hand-vectorised kernel; no CKKS
CHEHAB-RL	learned	BFV, VecLang S-expr.	as CHEHAB: algebraic rewrites of a fixed exact circuit
Fhelipe	analytical, ML	CKKS, tensor DSL	none: activations are fixed polynomial templates supplied by the user
EinHops	analytical	CKKS, einsum	none: linear algebra only, no non-polynomial ops
CHET	rule-based, ML	CKKS, tensor DSL	none: activations must be supplied as polynomials
EVA	rule-based	CKKS, vector DSL	none: scale management of a user-written circuit
HECO	rule-based	BFV/BGV, MLIR	none: exact arithmetic, batching rewrites of a fixed circuit
Hecate	rule-based	CKKS, scale optimisation	none: rescaling placement of a user-written circuit
Porcupine	synthesis	BFV, Quill kernels	none: exact integer kernels; BFV-matvec would be expressible but its output is a schedule, not an algorithm
Coyote	search-based	BFV, arithmetic circuits	none: exact vectorisation of a fixed circuit

I Component Ablations on All Workloads

Table 10 extends Table 3 to the six workloads with ablation runs, single seed. On GeLU, Sigmoid, Eth-Fraud, and Sign the seeds sit at or near the depth ceiling and the columns scatter within ± 0.01 of the full system on both sides, which is single-seed run-to-run variance; Sign additionally shows session-level eval-time drift of about 0.2 s on identical seed programs, which exceeds any component effect.

Table 10: Single-seed component ablations, combined score. "Full" is the full system in the ablation batch; "Vanilla" the three-seed mean from Table 1.

Workload	Full	−Spec	−Strategy	−Hier	−Repair	Non-evo	Vanilla
Softmax	**0.738**	0.702	0.706	0.707	0.699	0.722	0.702
ReLU	0.862	0.866	0.857	**0.877**	0.826	0.800	0.851
Eth-Fraud	0.952	**0.953**	0.945	0.943	0.950	0.943	0.891
Sign	0.746	0.739	**0.754**	0.743	0.746	–	0.658
GeLU	0.512	**0.528**	0.526	0.516	0.520	0.517	0.523
Sigmoid	**0.893**	0.891	0.892	**0.893**	0.891	–	0.895

J Certified Approximation Bounds

Correctness object. FHEvolve is heuristic search with empirical validation, not verified synthesis. The right correctness object differs from that of a compiler. A compiler over exact arithmetic can prove that a rewrite is *equivalent* to its input. An approximate CKKS program has no equivalence to prove: its output is a polynomial approximation of a non-polynomial target, and the certifiable statement is a sup-norm bound on the approximation error over the input range. The end-to-end error of an evolved program decomposes into three terms. (i) The *program semantics*: the evolved code composes OpenFHE primitives whose semantics the library fixes, so the only free semantics is the arithmetic expression, checked slot-wise against the plaintext reference on every evaluation.

(ii) The *approximation error* $|p(x) - f(x)|$, a property of the mathematics, which we certify over the whole observed input range with a Lipschitz bound on a $2{\times}10^7$-point grid: GeLU $< 8.18{\times}10^{-4}$ on $[-6.79, 7.06]$ and Sigmoid $< 9.40{\times}10^{-4}$ on $[-25, 25]$. (iii) The ***CKKS noise***, bounded by scheme analysis. Composite pipelines reduce to single-polynomial bounds by Lipschitz composition: the evolved ReLU is $\frac{1}{2}(x + s(x^2))$ with s a Chebyshev fit of $\sqrt{\cdot}$, so $|\mathrm{ReLU}_p(x) - \mathrm{ReLU}(x)| \le \frac{1}{2}\,\varepsilon_s$ where ε_s is the certified bound of s on $[0, 1]$, giving $< 2.6{\times}10^{-3}$ for the evolved program. Softmax composes the same way given a range bound on the denominator, which we leave for future work. Bounds hold on the fitted range only .

Certification method. The error of an evolved FHE polynomial program decomposes into (1) the approximation error $|p(x) - f(x)|$, a property of the mathematics independent of FHE, and (2) CKKS evaluation noise, bounded by standard scheme-level analysis. Empirical testing conflates the two and checks only sample points. We certify (1) over the whole input range with a Lipschitz bound: evaluate $e(x) = p(x) - f(x)$ on a $2{\times}10^7$-point grid and bound the between-grid excursion with a certified bound on $|e'(x)|$ (correction $\approx 3{\times}10^{-9}$). Naive interval arithmetic gives a useless bound of 0.24 because of the dependency problem.

Table 11: Certified sup-norm bounds on evolved single-polynomial solutions. Coefficients are those OpenFHE evaluates, not reconstructions.

Workload	**Domain**	**Certified bound**	**Within** 10^{-3}
GeLU (deg 24)	observed input range $[-6.79, 7.06]$	$8.18{\times}10^{-4}$	yes
GeLU (deg 24)	0.1 outside the range	$2.26{\times}10^{-3}$	no
GeLU (deg 24)	0.5 outside the range	0.568	no
Sigmoid (deg 51)	fit range $[-25, 25]$	$9.40{\times}10^{-4}$	yes
Sigmoid (deg 51)	0.5 outside the range	3.69	no

The bound holds for every x in the observed input range, not only at the tested points, but it is tightly domain-restricted: the polynomial has no safety margin beyond the fitted range. A deployment claim must pin the input range.

Composition. Composite pipelines reduce to single-polynomial bounds when each stage is Lipschitz on the range of the previous one. The evolved ReLU computes $\frac{1}{2}(x + s(x^2))$ with s a degree-47 Chebyshev fit of $\sqrt{\cdot}$ on $[0, 1]$. Since $x^2 \in [0, 1]$ for $x \in [-1, 1]$ and $\mathrm{ReLU}(x) = \frac{1}{2}(x + \sqrt{x^2})$ exactly, $|\mathrm{ReLU}_p(x) - \mathrm{ReLU}(x)| = \frac{1}{2}|s(x^2) - \sqrt{x^2}| \le \frac{1}{2}\varepsilon_s$, where ε_s is the certified sup-norm bound of s on $[0, 1]$ computed by the same grid procedure. With $\varepsilon_s = 5.2{\times}10^{-3}$ the evolved ReLU is certified to $2.6{\times}10^{-3}$, which is consistent with its measured 98.97% slot accuracy at the 10^{-3} tolerance: the bound is tight near $x = 0$ where the square root is not Lipschitz, and the tolerance is violated only there. Softmax composes $\exp$ (iterated squaring), a Chebyshev reciprocal, and a Newton step; propagating the bound requires a lower bound on the encrypted denominator, which the spec's input range provides, and is left for future work.

K Compute-Efficiency and Convergence

Table 12 reports, per workload, the iteration at which each method's best score was found, the final candidate's eval time, and the resulting combined score. On the high-headroom workloads (Softmax, ReLU, Eth-Fraud), FHEvolve continues to improve late (best at iteration 191 / 187 / 195 of 200), whereas Vanilla's best is fixed by iteration 174 (Softmax) and 84 (ReLU). On the depth-saturated workloads (Sigmoid, GeLU) the two methods converge to within $\sim$0.003, but FHEvolve reaches its best earlier on Sigmoid (iteration 114 vs. 190). The non-evolutionary variant of §5.5 plateaus between iteration 9 and 85 on every workload (Softmax seeds: 9, 32, 27).

L Token Usage and API Cost

Token counts come from tokenizing the logged prompts with the `o200k_base` encoding as a proxy for the provider tokenizer. Across 856 logged iterations FHEvolve averages about 10,800 input and

Table 12: Compute efficiency: iteration of best score, eval time per candidate, and final combined score (original three-seed batch, representative seed).

Workload	Method	Best at iter	Eval time	Final score
Softmax	Vanilla OE	174	8.48 s	0.7021
	FHEvolve (ours)	**191**	**7.11 s**	**0.7376**
GeLU	Vanilla OE	61	∼4.6 s	0.5220
	FHEvolve (ours)	140	4.53 s	0.5244
ReLU	Vanilla OE	84	0.87 s	0.8459
	FHEvolve (ours)	187	**0.57 s**	**0.8789**
Sigmoid	Vanilla OE	190	0.34 s	**0.8982**
	FHEvolve (ours)	**114**	0.35 s	0.8952
Eth-Fraud	Best seed	0	0.61 s	0.8910
	FHEvolve (ours)	**195**	**0.17 s**	**0.9675**

710 output tokens per iteration (Table 13), so a 200-iteration run uses roughly 2.16 M input and 0.14 M output tokens, about USD 4.10 at the gpt-5 rate of USD 1.25 per million input and USD 10 per million output tokens. The FHEvolve-specific part, the spec block and strategy menu in the system message, is about 2,155 tokens per call, roughly 20% of the input, or about USD 0.54 per run. This scaffolding is identical on every call, so prompt caching reduces it further. Wall-clock time is dominated by OpenFHE evaluation, not LM latency (Table 12). A matched Vanilla token count was not logged.

Table 13: FHEvolve token usage per iteration, averaged over logged prompts.

Workload	Iters logged	Input tok	Output tok	Scaffolding tok
GeLU	116	11,605	982	2,362
ReLU	109	8,275	591	2,252
Sigmoid	89	7,964	584	2,307
Sign	180	9,953	886	2,193
Softmax	128	13,865	608	2,957
Eth-Fraud	129	15,081	576	2,446
BFV-matvec	105	7,540	636	300
Overall	856	10,818	711	2,155

M Static Depth-Audit Robustness

We ran the audit on 917 unique logged candidates across seven workloads. It never failed to parse, and it rejected none of the 667 candidates that OpenFHE evaluated cleanly (zero false positives). Across 1,410 evaluations there were no depth failures, because the depth-aware scaffolding kept every candidate feasible, so the audit acted as a never-triggered safety net and its false-negative rate cannot be measured from the natural candidate stream. We therefore stress-tested it on synthetic programs with known depth (Table 14).

Table 14: Static depth audit on synthetic code structures with known multiplicative depth.

Structure	True depth	Audit estimate	Result
Standard Chebyshev, degree 27	6	6	exact
Degree 120 hidden in a helper function	8	8	exact
Commented-out degree-500 decoy	6	6	exact (ignores comment)
Dynamically computed degree (30+90)	8	0	under-estimate
Loop-based iterated squaring, K=15	15	1	under-estimate
Unrolled Horner, 20 ct×ct mults	20	5	under-estimate

Every error is an under-estimate, so a deep candidate reaches the evaluator and costs one wasted run. We observed no false rejection among the 667 successfully evaluated candidates; that sample does not establish the audit's broader reliability.

N FHEvolve Component Details

Design rationale. Each component rests on a property of FHE that makes its job tractable. *Spec Compiler:* the relevant constraints (Paterson–Stockmeyer depth costs, rotation-stride composability) are formal and universal across CKKS workloads, so the compiler can choose the budget and leave the algorithm to the LM; the same separation generalises to other schemes. *Strategy Library:* FHE algorithm patterns live on a small Pareto frontier of depth $\times$ rotation count $\times$ accuracy class, and the same pattern instantiates across workloads at the same op-type, so the catalogue can be typed and each entry's depth cost known statically. *Hierarchical Evolver:* a strategy swap can change the candidate's depth class while a parameter tweak cannot, and this decomposition recurs in BFV with discrete depth tiers. *Repair Classifier:* FHE failures fall into a small, well-typed set of modes whose regex fingerprints are stable across OpenFHE versions, and the same depth arithmetic the Spec Compiler uses can be turned adversarially to detect doomed candidates.

Spec schema. The `Spec` record carries: *crypto parameters* (scheme tag, multiplicative-depth budget, ring dimension, scaling-modulus size, rotation-key set, batch size, bootstrapping flag); *workload shape* (input/output ranges, per-slot accuracy threshold); *op-type tag* (`elementwise_polynomial`, `softmax_composite`, `comparison_reduction`, `linear_algebra`, ...); and *derived constraints* (the largest single-call Chebyshev degree fitting `mult_depth`-2 under Paterson–Stockmeyer, and the shortest two-hop decompositions of common power-of-two rotation strides through the provisioned key set).

Behaviour outside the library. An unrecognised challenge is assigned the default op-type `elementwise_polynomial` and receives the generic activation menu. A recognised op-type with no matching entry (e.g. `lookup_table`, `bitwise`) produces an empty menu; the framework then runs on the extracted spec and the hierarchical prompt alone, which amounts to generic guided search with the depth audit and typed repair still active. The $-$STRATEGY column of Table 3 is the empirical proxy for this condition.

Strategy Library entries. Thirteen op-type-typed entries: Chebyshev direct fit, $|x|$ decomposition, $\tanh$ decomposition, iterated-squaring $\exp$ with range reduction, log-depth sum reduction, mask-and-broadcast, Chebyshev $1/x$ with optional Newton–Raphson refinement, the full Softmax pipeline, smooth pairwise tree max, sign-gated max, soft-max argmax aggregator, composite-minimax sign, and stride-N Hadamard+rotate-and-sum matmul. Each bundles a parametrised OpenFHE code template, a "when to use" heuristic, tunable parameters, and three FHE-typed metadata fields (depth tier, minimum depth budget, supported schemes).

Repair-classifier buckets and hints. The classifier maps every failed traceback into one of eight typed buckets (Table 15), each carrying a one-line repair hint that flows back to the LM in the next mutation prompt.

Table 15: Crypto-typed repair buckets and the one-line hints attached to each.

Bucket	Repair hint
`depth_exhausted`	lower Chebyshev degree, reduce iterated-squaring K, drop one Newton iter, fold ct$\times$ct $\rightarrow$ ct$\times$pt
`rotation_key_missing`	only rotate by indices in `spec.rotation_keys`; recompose unsupported strides via two hops
`decrypt_error`	check the output ciphertext is at the expected level/scale; rescale before `Decrypt`
`timeout`	reduce poly degree, drop a Newton iter, or fall back to a lower-tier strategy
`accuracy_collapse`	widen the Chebyshev fit interval, shorten range-reduction K, or switch to a less aggressive decomposition
`context_error`	use only crypto parameters declared in `spec`; do not redefine the context inside `solve`
`import_error`	restrict imports to OpenFHE-Python and the project standard set
`runtime_error`	inspect the original traceback below; the failure is generic Python, not crypto-typed

Static depth audit. The pre-evaluation audit parses the candidate source for declared Chebyshev degrees, iterated-squaring counts, and Newton-iteration counts, sums their depth costs under the same

Paterson–Stockmeyer rule the Spec Compiler uses, and rejects the candidate before OpenFHE is invoked when the sum exceeds `spec.mult_depth` -2 (the two-level margin absorbs rescaling slack).