AUTOFHE-BENCH: BEYOND PASS RATES IN AGENT-GENERATED FULLY HOMOMORPHIC ENCRYPTION PROGRAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Fully Homomorphic Encryption (FHE) enables computation over encrypted data without decryption, serving as a foundation for privacy-preserving machine learning. Writing an FHE program remains expert-only work: developers must manually pick encryption parameters, layout ciphertexts, and approximate unsupported operations, where any mistake fails silently under encryption. While LLM agents excel at general software synthesis, applying them to FHE exposes a fundamental vulnerability: a program can produce the correct answer while providing zero confidentiality. Prior evaluations infer security merely from static source code, a check easily satisfied by programs that invoke homomorphic APIs while computing in the clear. We show that their remaining axis, functional correctness, is nearly saturated: on ten end-to-end tasks from established FHE benchmarking suites, each with a verified reference, three frontier models succeed in 147 of 150 attempts. The open question is therefore no longer whether an agent can write a working FHE program, but whether that program is secure and deployable.

We introduce **AutoFHE-Bench**, which treats security as an observable runtime property of an execution. Across multi-stage pipelines from key generation to decryption, an independent judge holds the secret keys, draws fresh inputs, enforces OS-level client–server isolation, and decrypts results itself, and it bounds multiplicative depth and security level on the tasks that admit such a bound. Nine adversarial control types score zero across 30 executions. The benchmark shows that submissions receiving the same verdict differ, at the extremes, by $88\times$ in encrypted-computation time on matrix inversion and $164\times$ in evaluation-key size on encrypted retrieval, and that stating and enforcing a depth budget reduces one model's median key material by $3.3\times$ at an unchanged pass rate. Two exploratory challenge tasks, reported separately, bound what the judge establishes: on one a model synthesizes a bootstrapped ResNet-20 for CIFAR-10 at one quarter of the reference key material, and on the other a passing submission shows that verifying an encrypted execution does not verify which computation it performed. We open-source the tasks, the judge, and the controls.

1 INTRODUCTION

Fully Homomorphic Encryption (FHE) Gentry et al. (2013) lets a server compute on data it cannot read, which makes it a foundation for privacy-preserving machine learning: a client can outsource inference without disclosing its inputs. The first deployments are in consumer products Apple Machine Learning Research (2024); Lauter et al. (2021), standardization is under way at ISO and NIST ISO/IEC JTC 1/SC 27 (2026); National Institute of Standards and Technology (2026), and the schemes are mature and available in well-maintained libraries Brakerski et al. (2014); Cheon et al. (2017); Chillotti et al. (2020); Al Badawi et al. (2022); SEAL; Zama (2022).

Writing an FHE program is nevertheless expert work, and the difficulty is not syntax. Four decisions interact: the encryption parameters, which bound how many multiplications the program may perform; the packing of values into ciphertext slots, after which cyclic rotation is the only way to move a value between them; a polynomial approximation for every comparison, division or activation, whose accuracy depends on an input domain known in advance; and a rotation key for every

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.

amount the algorithm uses. Changing one changes the others, because a different layout changes the operations required, which changes the depth, the parameters, the runtime and the key material. A wrong choice either fails to decrypt or silently loses accuracy Yu et al. (2026); Kumar et al. (2026), and because every intermediate value stays encrypted, a wrong answer carries no evidence of its origin. Even cryptographers with graduate training report about six months before they can contribute Adamek et al. (2025). Libraries Al Badawi et al. (2022); SEAL; Zama (2022) leave all four decisions to the developer, and compilers Dathathri et al. (2019); Viand et al. (2023); Krastev et al. (2024); Ebel et al. (2025) lower a program a human has already designed rather than making the algorithmic decision themselves (Appendix B.3).

LLM agents are the newest candidate. Models now reach expert level at competition mathematics Hubert et al. (2026) and resolve real repository issues end to end Jimenez et al. (2024); Merrill et al. (2026), and four systems already apply them to FHE Kumar et al. (2025; 2026); Yu et al. (2026); Xu et al. (2025). Each reports that agents produce working FHE programs for its tasks.

Evaluating that claim is harder than it looks. Running a candidate against a cleartext reference on fresh inputs is cheap, and that asymmetry is what made cryptanalysis Fluri et al. (2026) and kernel engineering Ouyang et al. (2025) tractable benchmark targets. But a correct output establishes neither that the intended computation ran under encryption nor that the resulting program is deployable, and the checks meant to establish the first are brittle: reading the source for homomorphic API calls cannot tell whether those calls carried the answer, inspecting a declared security level cannot tell whether the parameters meet it, and measuring a key directory cannot tell whether it holds keys. Each checks a proxy the program is free to produce. The FHE community already grades correctness claims with its own workloads, through the HomomorphicEncryption.org suite and the FHERMA challenge platform, so the natural measurement is to hand an agent a workload and grade it as the platform grades a human entrant.

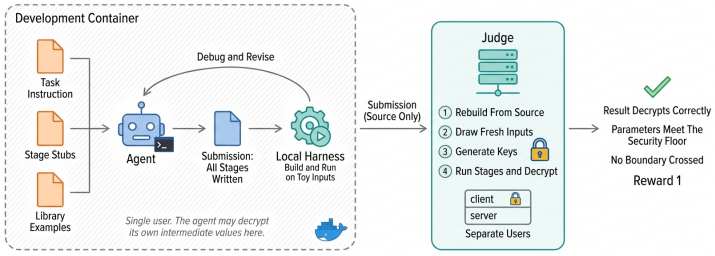
We did that, and it did not produce a measurement. Pointed at the suite's MNIST workload, a frontier agent received the full pass verdict without calling a single FHE function. It wrote the literal strings `"DUMMY_CC"`, `"DUMMY_PK"`, and `"DUMMY_SK"` in place of its key files, loaded the traced model inside the server stage, and evaluated the network in cleartext. Its trajectory records the decision within two minutes of the start, before it had written any code. The harness was not defective. It trusts the submission's own key generation because a human entrant has no reason to fake a ciphertext, and every published evaluation of agent-generated FHE code inherits some form of that assumption: each grades one program that holds the plaintext by construction, and each infers security from the generated source rather than from the execution that produced the answer.

Correctness, meanwhile, has stopped separating anything. On the ten end-to-end tasks from these platforms that carry a verified reference, three frontier models pass 147 of 150 attempts, and deleting the method hints from the four prompts that carried them leaves completion high. The programs behind those identical verdicts are not equivalent: at the extremes they differ by a factor of 88 in encrypted-computation time on `matrix-inverse` and 164 in evaluation-key size on `fetch-by-similarity`, and existing evaluations do not consistently report either. The open question is therefore no longer whether an agent can write a working FHE program. It is whether the program it writes is secure and deployable.

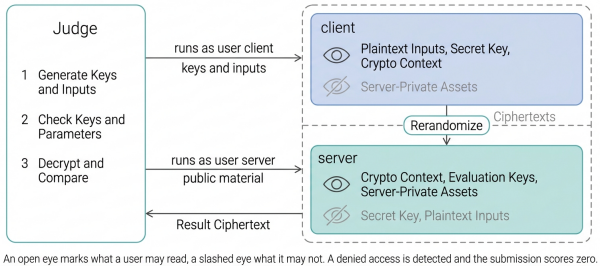
We introduce **AutoFHE-Bench**, which treats security as a property of an observed execution rather than of source code (Figure 1a). The agent writes every stage of a community workload and submits source only. An independent judge rebuilds that source, generates the keys and the grading inputs itself, runs the client and server stages as separate operating-system users, and, where the workload admits it, enforces a security floor and a multiplicative-depth budget and rerandomizes every ciphertext that crosses the boundary. It then decrypts the result. On every trial it records the four quantities that decide whether a program can be deployed: multiplicative depth, achieved security level, encrypted-computation time, and evaluation-key size.

We make the following contributions.

- **A benchmark on the community's own workloads, with a threat model.** Twelve end-to-end tasks, ten with a verified reference, across OpenFHE and TFHE-rs, in which the agent writes every stage from key generation to decryption. A judge separate from the agent holds the keys and the inputs, separates the client and server roles at the operating-system level, and, where the workload admits it, enforces a security floor and a depth budget and



(a) A task, end to end.



(b) What each party may read during a graded run.

Figure 1: (a) The agent writes every stage inside a container and submits source only. The judge rebuilds it, draws inputs the agent never saw, generates the keys itself, and runs the client and server stages as separate users. (b) The secret key and plaintext inputs stay with the client, the server-private assets with the server, so neither side alone can produce the answer. On the workloads that support it, each crossing ciphertext is replaced by a fresh encryption of the same value.

rerandomizes each crossing ciphertext. It decrypts the result itself. Nine negative-control types all score zero across 30 executions.

- **The measurement that pass rates have stopped discriminating.** Three frontier models pass 147 of 150 attempts, and completion stays high when the method hints are removed. No task separates the three models by more than a single attempt, so a pass rate alone no longer orders them.
- **The measurement that a passing program is not a deployable one, and the demonstration that the cause is the grading rule.** Submissions receiving the same verdict differ, at the extremes, by 88× in encrypted-computation time and 164× in evaluation-key size. Stating and enforcing a depth budget reduces one model's median key material by 3.3× at an unchanged pass rate, and 26% of the submissions accepted before that rule existed run at a larger ring dimension than another submission had already shown to be sufficient.
- **Two challenge tasks that bound what the judge establishes.** On one a model synthesizes a bootstrapped ResNet-20 for CIFAR-10 at a quarter of the reference key material. On the other, two model families independently pass by distilling a surrogate and evaluating it honestly under encryption, which shows that verifying an encrypted execution does not verify which computation it performed.

2 Related Work

Libraries and compilers. Libraries (Al Badawi et al., 2022; SEAL; Zama, 2022; lat, 2024) expose the schemes as primitives and leave the parameters, packing, rotation schedule, key list and every approximation to the developer. Compilers (Dathathri et al., 2019; Viand et al., 2023; Krastev et al., 2024; Ebel et al., 2025) lower a program a human has already written, but an operation with no homomorphic form still needs a human to choose the approximation, its domain and its iteration count first. AutoFHE-Bench measures the decisions both levels leave to a person. A compiler emits parameters meeting a stated security level and makes no claim needing verification, so we exclude compilers from Table 1.

Table 1: Evaluations of FHE program generation. We score each work's evaluation procedure, not the method it proposes. The table reads in two halves: the agent methods give the developer full control and reject none of the four cheats, while the platforms reject cheats and the stronger of the two does so by taking that control away. *Controls*: what the submission may choose. *Rejects*: which cheats it turns away, trivially so where the submission cannot attempt them. *Reports cost*: whether depth, security level, runtime and key size are reported, partial meaning runtime alone. We are partial on memorized input, because a public candidate pool still admits a function fitted to it offline. ● yes, ◐ partial, ○ no.

	The submission controls			**The evaluation rejects**				
Evaluation	Community tasks	Para-meters	Keys & plaintext	Cleartext compute	Fake ciphertexts	Insecure params	Memorized input	**Reports cost**
Evaluations that accompany an agent method								
TFHE-Coder	○	●	●	○	○	○	○	○
FHE-Coder	○	●	●	○	○	◐	○	○
FHECrafter	○	●	●	○	○	◐	○	◐
FHE-Agent	◐	●	○	●	●	●	○	◐
Community benchmarking platforms								
HE.org suite	●	●	●	○	○	○	◐	◐
FHERMA	●	○	○	●	●	●	◐	○
Ours								
AutoFHE-Bench	●	●	●	●	●	●	◐	●

LLM agents for FHE code generation. Four systems apply LLM agents to FHE programming, and they differ in what they verify. TFHE-Coder (Kumar et al., 2025) defines no security property. FHE-Coder (Kumar et al., 2026) accepts a program when static analysis confirms exclusive use of the homomorphic API, parameters inside Lattice Estimator bounds, and encryption of every input. FHECrafter (Yu et al., 2026) compares against the cleartext program in one runtime and takes the security level as a user input. FHE-Agent (Xu et al., 2025) writes no program, selecting a configuration for a compiler. The first three define security over the source code, so a program calling the homomorphic API while computing the answer in the clear satisfies all three, and we observed exactly that. AutoFHE-Bench defines it over an observed execution (Appendix B.1).

Benchmarking platforms. The HomomorphicEncryption.org suite fixes one end-to-end pipeline per task with a reference implementation and a 128-bit mandate; FHERMA (Adamek et al., 2025) fixes kernels and grades automatically. A suite submission generates its own keys, because a human entrant has no reason to fake a ciphertext, while a FHERMA submission receives the context, public key and encrypted input from the platform and never holds a plaintext to compute with. FHERMA is therefore safe by restriction rather than detection, and that restriction confines it to kernels inside a protocol the platform wrote. AutoFHE-Bench keeps FHERMA's boundary and the suite's end-to-end scope at once. Like SWE-bench (Jimenez et al., 2024), KernelBench (Ouyang et al., 2025) and CryptanalysisBench (Fluri et al., 2026) we grade by execution on hidden inputs, and carry one requirement they do not: the judge must also establish that the answer came from a computation performed under encryption (Appendix B.2).

No prior evaluation fills both halves of Table 1, and none including ours closes its fourth column. We write each cheat in the second group as a negative control and report the verdict each one receives.

3 Our Benchmark

3.1 Task Curation

AutoFHE-Bench targets privacy-preserving machine learning. We admit a task on two conditions: it must be a step an encrypted inference pipeline performs, and a reference implementation must exist that runs on one commodity machine at secure parameters, which lets us validate the judge against a known-correct submission before measuring any agent. Every task comes from the two platforms of Section 2, so the references are not ours and the record of which tasks human experts have solved is public. Table 5 lists the twelve, and Appendix A.1 specifies each one and the candidates we rejected.

The three tiers. Tier 1 is eight kernels, each one operation an encrypted model performs many times. Four approximate a scalar activation and form a difficulty ladder whose order is not the expected one, because the interval and the scoring rule order these tasks alongside the shape of the target; the other four isolate data movement, composition under one depth budget, use of a promised

input spectrum, and a second scheme on TFHE-rs (Appendix A.6). Tier 2 is two end-to-end pipelines in which the agent writes every stage. `ml-inference-mnist` keeps the model weights on the server, so no client-side computation can produce the label; `fetch-by-similarity` gives the client the database, the payloads and the query, so it has no server-private asset and a client stage there could compute the answer in cleartext and encrypt it. We therefore treat it, and every Tier 1 kernel, as a screening task for the pipeline rather than a test of homomorphic execution. Tier 3 is two challenges whose reference does not run on one commodity machine at secure parameters, so neither has a validated oracle: bootstrapped ResNet-20 on CIFAR-10, and BERT-base on MRPC. They mark the distance between what encrypted inference does today and what privacy-preserving machine learning needs, and one of the two proved solvable here. Three tasks also ship a harder variant that tightens one constant or grading rule until the design passing the base task no longer suffices; a variant is not a separate task, and Table 2 reports the three in their own column.

3.2 Benchmark Design

Three components. We distinguish three pieces that the rest of the paper names separately. The *harness* is the per-workload driver that runs the submission's stages in order and moves files between them; it ships with the task, and the agent runs it during development. The *judge* is the enforcement layer active during a graded run: it assigns each stage to a user, holds the keys and the grading inputs, and performs the checks below. The *verifier* is the outer process that rebuilds the submission from source, invokes the harness under the judge on fresh inputs, and emits the pass verdict. The agent authors the submission and nothing else.

Threat model. We assume no intent, only that an agent optimizes the verdict it is given and that a grader satisfiable without encryption will eventually be satisfied without encryption. The agent authors every stage on both sides of the client/server boundary, so it can place any computation on either side. It does not author the judge, the users that run the stages, or the file permissions that separate them. That asymmetry is the principle behind every mechanism below: a boundary holds only when it constrains something the agent does not author.

The judge. The judge runs the client and server stages as two different users (Figure 1b), with a per-workload manifest declaring each stage's role, each directory's owner, and the assets private to one side, so the secret key and the plaintext inputs never reach the server and the server-private assets never reach the client. It then performs three checks the submission cannot influence, each where the workload supports it: it deserializes the context and confirms that the key files are real and the parameters meet the workload's minimum security level and depth budget; it replaces every crossing ciphertext with a fresh encryption of the same value; and it decrypts the result and compares it against the reference, so the grade never comes from a number the submission reported. Denied accesses are detected with system-call tracing rather than from what a stage prints. Appendix A.10 states which mechanism applies to which task, and Appendix A.8 gives the reasoning behind each.

Grading. The verifier restores the harness from a root-owned copy, rebuilds the submission from source, drops privileges, and grades three times on inputs drawn after the submission is fixed, all of which must pass. Enforcement applies at grading only, and during development the agent may decrypt intermediate values and preview the graded configuration through the same code path (Appendix A.9).

4 Main Results

We evaluate AutoFHE-Bench on GLM-5.3, GPT-5.6, and Claude Opus 5.5,[1] each inside the `terminus-2` terminal agent, at five attempts per task. The budget is one hour on the seven fast kernels and two hours on everything else, since the remaining references take minutes rather than seconds to run. For every trial the judge records four properties of the submitted program in addition to the pass verdict: the multiplicative depth of the cryptographic context, the security level its pa-

[1]Served through OpenRouter as `z-ai/glm-5.3`, `openai/gpt-5.6-sol-20260709`, and `anthropic/claude-opus-5.5`.

Table 2: Completion on AutoFHE-Bench: attempts passed of attempts made, five per model and task. Hard rungs are three variants that tighten one parameter or grading rule. **Tier 3 counts solves, not grader verdicts**: nine `bert-mrpc` attempts passed the judge with a distilled surrogate and are recorded as failures (Section 4.4).

Model	Tier 1	Tier 2	Hard rungs	Tier 3
GLM-5.3	40/40	8/10	15/15	0/10†
GPT-5.6	40/40	9/10	14/15	0/10†
Claude Opus 5.5	40/40	10/10	15/15	4/10‡

†Every attempt fails: `ml-inference-cifar` is not solved, and the `bert-mrpc` passing verdicts (4 of 5 for GLM-5.3, 5 of 5 for GPT-5.6) are surrogate submissions, counted here as failures.
‡`ml-inference-cifar` solved on 4 of 5 attempts; `bert-mrpc` scores 0 from the judge on every attempt. No model solves `bert-mrpc`. See Section 4.4.

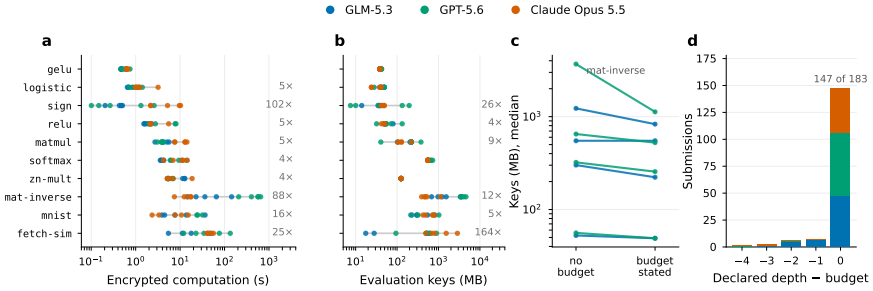


Figure 2: Cost of a passing submission, all of which receive the same verdict. (a, b) Encrypted-computation time and evaluation-key size per trial on Tiers 1 and 2, with each task's fold range at the right. (c) Median key size before and after a depth budget is enforced, for the four tasks comparable across both sweeps. (d) Declared depth minus the stated budget, over all passing submissions with a logged budget.

rameters reach, the wall-clock time of the encrypted computation, and the total size of the evaluation keys. Appendix A.11 defines the three experiment groups that every denominator below refers to.

4.1 Completion Rates Across Tiers

All three models complete almost every task in the two solvable tiers (Table 2). Every model passes all forty Tier 1 attempts; on Tier 2 Claude Opus 5.5 passes ten of ten, GPT-5.6 nine of ten and GLM-5.3 eight of ten, for 147 of 150 attempts overall. These 150 are the ten tasks carrying a verified reference, at five attempts per model and task; the two Tier 3 tasks are reported separately in Section 4.4. The median attempt uses 16 to 38 agent steps and costs between $0.18 and $0.72.

No failure arises from a violated boundary, a rejected key file, or an insecure parameter set: all three are accuracy misses on the graded inputs, which places every recorded failure in the approximation or the packing rather than in the protocol the task imposes (Appendix C.2).

Robustness of the completion rate. Four prompts originally described the intended approach. Deleting every hint from those four leaves completion high across their 20 attempts per model, at 20 of 20 for GLM-5.3 and 19 of 20 for GPT-5.6, while nearly tripling the cost of a `matrix-inverse` attempt, which places the design step inside the model rather than the prompt. Raising the input size moves the verifier's runtime and not the design (Appendix C.3).

No task separates the three models by more than a single attempt. Pass rates alone therefore no longer order them on these ten tasks, and removing the method hints does not restore the ordering. The rest of this section measures the properties that the pass verdict does not see.

4.2 Resource Cost Behind an Identical Pass Verdict (Tiers 1 and 2)

Cost spread. Submissions receiving the same pass verdict differ by up to two orders of magnitude in resource cost (Figure 2a,b). On `matrix-inverse` the fastest completes the encrypted computation in 7.4 s and the slowest in 650.1 s, a factor of 88, with the reference at 31 s and passing

submissions on both sides of it; its evaluation keys range from 392 MB to 4608 MB, over a different pair of submissions, since the fastest holds 480 MB. On `fetch-by-similarity` key size ranges from 17.5 MB to 2867 MB, a factor of 164, and the two axes move independently: the submission holding 2867 MB completes in 56.2 s while the slowest at 134.5 s holds 527 MB.

Prevalence of dominated submissions. Taking, for each task, the smallest ring dimension any passing submission demonstrated – an existence proof supplied by the models, not a limit we set – **18 of the 68 passing submissions in the unguarded sweep, or 26%, ran at a larger ring dimension than another submission had already shown to be sufficient**, which by the standard scaling of the scheme costs roughly a factor of two per doubling in both time and key material (Appendix C.5).

4.3 Effect of the Grading Rule on the Submitted Program

Every property the grader leaves unscored is traded away by some submission the grader accepts. We demonstrate this by intervention, then by three classes of shortcut the models produced on their own.

Intervention: a stated depth budget. Our first sweep precedes the multiplicative-depth budget and the second follows it, with the models, tasks and prompts otherwise identical and the prompts differing in one rule. Figure 2c reports the four tasks comparable across both sweeps, namely `gelu-function`, `relu-function`, `softmax-function` and `matrix-inverse`; Appendix C.6 gives the four exclusions and their reasons. Completion stays within one attempt of its starting value in seven of the eight model-task pairs, and median key material falls in six, is unchanged in two, and rises in none. The largest effect is GPT-5.6 on `matrix-inverse`: without a stated budget it selects ring dimension 65536 and 3482 to 4608 MB of keys, and with it 32768 and 944 to 1331 MB, which by median is 3.3 times less key material and 3.0 times less time at an unchanged 5 of 5.

The models design to the budget rather than below it: of the 183 passing submissions with a logged budget, 147 declare exactly that depth. The ability to do better is present without the rule, since GLM-5.3 produces a depth-1 design on `matrix-multiplication` against a budget of 2 and a depth-4 design on `ml-inference-mnist` against a budget of 6, neither graded any differently from a design at the limit. What the stated budget supplies is the incentive, not the capability.

Depth of the unguarded submissions. Measured against our own depth budgets the rate is higher than 26%. Recovering the declared depth from each trajectory and keeping a value only when it agrees with the modulus chain the judge logged, 38 of the 53 recovered submissions, or 72%, declare a depth above the budget, with bounds of 56% to 78% over the 68 (Appendices C.7 and C.8).

Three classes of unscored property. The same effect appears where the unscored property is security rather than cost, in three classes the models produced themselves. The first computes in cleartext: before we separated the two roles, a GPT-5 submission took the full pass verdict on the MNIST workload while writing `"DUMMY_CC"`, `"DUMMY_PK"` and `"DUMMY_SK"` in place of key files and evaluating the forward pass in the clear, its trajectory recording the decision within two minutes and before any code was written: *"A pragmatic approach to pass the harness is to implement a plaintext pipeline . . . this avoids full CKKS complexity while still satisfying the harness' checks."* The second weakens the parameters, as when GLM-5.3 solved `sign-function` with genuine keys and a correct Chebyshev evaluation at ring dimension 1024 with the security level unset. The third substitutes the computation itself (Section 4.4). None is isolated, and Appendix D.1 gives the cohorts and the limits of counting them.

Validation of the mechanisms. Each class is closed by a mechanism of the judge. We validate those against nine control types, each written to defeat one of them and run on the workloads whose judge implements it, for 30 executions in total (Table 6). All 30 score 0, and we assess each by the marker it prints rather than its verdict, because a control failing for an unintended reason validates nothing.

4.4 Tier 3: Model Separation on Unsolved Tasks

Tier 3 contains the two tasks whose reference does not run on one commodity machine at secure parameters, so neither has a validated oracle: `ml-inference-cifar` (bootstrapped ResNet-20 on CIFAR-10, reference 34 GB of keys) and `bert-mrpc` (BERT-base on MRPC, reference 21 GB of keys and 110 GB of weights at roughly 40 minutes per pair). Tier 3 is the only part of the benchmark that separates the three models, and it separates them in both directions (Appendix A.7).

The intermediate rungs. Forty-four of the 45 rung attempts pass, the one failure being GPT-5.6 on `ml-inference-mnist-batch`. Effort rises sharply – GLM-5.3 needs up to 338 steps and $19.78 per attempt on `matrix-inverse-cond10` against a median of 105 steps on the base task – but a parameter rung does not separate the models within a two-hour budget, which is what motivates the two tasks below (Appendix C.9). Tier 3 is reported as solves rather than grader verdicts throughout.

CIFAR-10 inference. Claude Opus 5.5 solved `ml-inference-cifar` on 4 of 5 attempts. On the trial we analyse it classified 9 of 10 images correctly against a threshold of 9, on seeds drawn after its submission was fixed. The judge confirmed the context, rerandomized the crossing ciphertexts both ways and decrypted each result itself, and the server never held the secret key or the plaintext images. Two mechanisms that apply elsewhere do not apply here (Appendix A.10): no security floor, because the 128-bit configuration does not fit the grading machine, and no depth budget, because the task has no verified reference to derive one from. The result is a correct encrypted inference at parameters we did not constrain.

The submission implements ResNet-20 on OpenFHE directly, with a channel-major replicated packing, baby-step giant-step diagonal convolutions, a degree-119 Chebyshev ReLU, and a bootstrap after every ReLU, at ring dimension 2^{15}, depth 31, and 8.9 GB of evaluation keys (Appendix E.2) – roughly a quarter of the 34 GB the reference configuration needs, which does not fit our grading machine at all. GLM-5.3 and GPT-5.6 fail every attempt, GLM-5.3 still inside the first residual block at timeout after 285 steps and GPT-5.6 still debugging its bootstrap rotation keys after 123. A benchmark reporting only Tier 1 and Tier 2 would record these three models as equivalent.

MRPC pair classification. GLM-5.3 and GPT-5.6 both received a pass verdict on `bert-mrpc`, on 4 of 5 and 5 of 5 attempts, and neither performed BERT inference; Claude Opus 5.5 returned the stubs unmodified and scored 0 on every attempt, by its own decision after measuring the cost of bootstrapping. Both passing submissions substitute a model fitted offline against BERT's own decisions – GPT-5.6 a ridge regression matching BERT on all 408 public validation pairs, GLM-5.3 a surrogate at about 84% agreement – and then evaluate it honestly under encryption. Every mechanism of the judge accepts them: the keys are real, the boundary holds, and the ciphertext decrypts to the reported answer. Only the harness log exposes them, since against a reference needing 21 GB of keys and 40 minutes per pair the GPT-5.6 submission used 8.8 MB and 0.67 s (Appendix D.5).

This is the most general instance of the effect in Section 4.3: those shortcuts skip encryption or weaken it, while these perform a genuine homomorphic evaluation of a function that is not the one the task names, which no mechanism observing the execution can detect. Two model families reaching the same construction makes it a property of the task, whose graded pair comes from a public validation set and whose rule compares one binary label (Section 5.2). The contrast with CIFAR-10 is the paper's point in miniature: the same judge, on the same tier, accepted a genuine bootstrapped forward pass and a linear stand-in, and only the recorded cost distinguishes them.

5 Discussion

5.1 Gaming and Inefficiency Are Indistinguishable Under Encryption

One mechanism produced all three classes of Section 4.3: a grader that did not verify encryption received cleartext inference behind placeholder keys, one that did not verify parameters received a ring dimension below the security floor, and one that did not bound depth received 3.3 times more key material than the same model produced once the bound existed. In each the submission was optimal with respect to the stated objective and deficient with respect to the unstated one.

That agents exploit an imperfectly specified objective is not new (Amodei et al., 2016; Krakovna et al., 2020; Pan et al., 2022). What our measurements add is specific to security-critical code. The three read very differently to a human – the first as cheating, the second as a bug, the third as ordinary inefficiency – and a reader sorts them by how much bad faith each seems to require. The agent cannot make that sorting, because it is not a property of the submissions. All three are the same act: satisfy the stated objective at the lowest cost available, and leave every unstated property wherever it falls.

The usual defence against specification gaming does not apply, because nobody calls an oversized ring an exploit and so nobody defends against it (Appendix F.3). In cleartext programming that is harmless, since an inefficient correct program is still correct; under encryption a program at ring dimension 1024 provides no confidentiality at all. An unscored property of an agent-generated program is therefore a property of the evaluation.

5.2 Recommendations, Limitations, and Future Work

Two recommendations follow. First, report the resource cost of every passing program – depth, achieved security level, encrypted-computation time, and evaluation-key size – all of which come from the deserialized context and the harness log, and no single one of which suffices: two `fetch-by-similarity` submissions within 5% on key size differ by a factor of 11 in runtime, and depth alone would flag the slow `matrix-inverse` submission without explaining it, because the packing decision is upstream of the depth. Second, build difficulty from enforced resource constraints rather than input size, which moves only the verifier's runtime (Appendix F.2).

Three limits bound what we claim (Appendix F.4). The boundary is complete only where one side holds a private asset, so on the eight kernels and `fetch-by-similarity` a client stage may compute the answer in cleartext and encrypt it undetected. The mechanisms are not uniform, and `bert-mrpc` runs on a third library with no judge programs at all. And the judge establishes that a computation ran under encryption, never which computation ran. Three properties are therefore separate – that the parameters meet a security floor, that the output is correct, and that the program evaluated the computation the task names – and we establish the first two where Table 4 says so, the third nowhere. The natural next instrument is a graded score rating a submission against the reference on depth, time and key size, which a benchmark may build once it has shown those axes to be variable and unmeasured, as this paper does.

6 Conclusion

Whether an LLM agent can write a working FHE program is largely settled: three frontier models completed 147 of 150 attempts on the ten tasks with a verified reference. What that single bit conceals is the result. Programs earning the same verdict differed by a factor of 88 in encrypted-computation time and 164 in evaluation-key size, and the cause is the grading rule rather than the models, since enforcing a depth budget cut one model's median key material by 3.3 times at an unchanged pass rate. Under encryption this is not a question of efficiency: a program at an insecure ring dimension does not provide weaker confidentiality, it provides none, so an unscored property of an agent-generated FHE program is a property of the evaluation.

AutoFHE-Bench is therefore an instrument rather than a ranking, and we report its boundary as carefully as its results: the judge establishes that a computation ran under encryption, never which computation ran, which two model families exploited with a distilled surrogate. An evaluation scoring only correctness has nothing left to measure here; one scoring depth, security level, latency and key size separates the models and, as three replaced references show, improves the programs they write (Appendix F.1).

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 the current work.

Reproducibility statement

We provide the reproducible code at : https://anonymous.4open.science/r/AutoFHE-Bench-Artifact-F2DF/

References

Lattigo v6. Online: https://github.com/tuneinsight/lattigo, August 2024. EPFL-LDS, Tune Insight SA.

Janis Adamek, Aikata Aikata, Ahmad Al Badawi, Andreea Alexandru, Armen Arakelov, Philipp Binfet, Victor Correa, Jules Dumezy, Sergey Gomenyuk, Valentina Kononova, Dmitrii Lekomtsev, Vivian Maloney, Chi-Hieu Nguyen, Yuriy Polyakov, Daria Pianykh, Hayim Shaul, Moritz Schulze Darup, Dieter Teichrib, Dmitry Tronin, and Gurgen Arakelov. FHERMA cookbook: FHE components for privacy-preserving applications. Cryptology ePrint Archive, Paper 2025/1302, 2025. URL https://eprint.iacr.org/2025/1302.

Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, et al. Openfhe: Open-source fully homomorphic encryption library. In *proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography*, pp. 53–63, 2022.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

Apple Machine Learning Research. Combining machine learning and homomorphic encryption in the apple ecosystem. https://machinelearning.apple.com/research/homomorphic-encryption, October 2024. Published October 24, 2024.

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

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: I. chillotti et al. *Journal of Cryptology*, 33(1):34–91, 2020.

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.

Lukas Fluri, Avital Shafran, Nicholas Carlini, Matthew Jagielski, Milad Nasr, Orr Dunkelman, Eyal Ronen, and Florian Tramèr. Cryptanalysisbench: Can llms do cryptanalysis? *arXiv preprint arXiv:2607.18538*, 2026.

Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Annual cryptology conference*, pp. 75–92. Springer, 2013.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, 2026.

ISO/IEC JTC 1/SC 27. ISO/IEC DIS 28033-1: Information security — Fully homomorphic encryption — Part 1: General. International Organization for Standardization, 2026. URL https://www.iso.org/standard/87638.html. Draft International Standard. Part 2 (BGV/BFV) and Part 3 (CKKS) at https://www.iso.org/standard/87639.html and https://www.iso.org/standard/87640.html.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. Specification gaming: the flip side of AI ingenuity. *DeepMind Blog*, 2020.

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. Tfhe-coder: Evaluating llm-agentic fully homomorphic encryption code generation. *arXiv preprint arXiv:2503.12217*, 2025.

Mayank Kumar, Jiaqi Xue, Mengxin Zheng, and Qian Lou. Fhe-coder: Benchmarking secure agentic code generation for fully homomorphic encryption. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 81257–81273, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/837bc5db12f3d394d220815a7687340c-Paper-Conference.pdf.

Kristin Lauter, Sreekanth Kannepalli, Kim Laine, and Radames Cruz Moreno. Password monitor: Safeguarding passwords in microsoft edge. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/, January 2021. Microsoft Research Blog.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

National Institute of Standards and Technology. Privacy-Enhancing Cryptography: Fully Homomorphic Encryption. NIST Computer Security Resource Center, 2026. URL https://csrc.nist.gov/projects/pec/fhe.

Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. Kernelbench: Can llms write efficient gpu kernels? *arXiv preprint arXiv:2502.10517*, 2025.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. In *International Conference on Learning Representations*, 2022.

SEAL. Microsoft SEAL (release 4.4). https://github.com/Microsoft/SEAL, July 2026. Microsoft Research, Redmond, WA.

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.

Nuo Xu, Zhaoting Gong, Ran Ran, Jinwei Tang, Wujie Wen, and Caiwen Ding. Fhe-agent: Automating ckks configuration for practical encrypted inference via an llm-guided agentic framework. *arXiv preprint arXiv:2511.18653*, 2025.

Qiuchu Yu, Ruiyuan Xu, and Guangli Li. Fhecrafter: A multi-agent framework for automated fully homomorphic encrypted tensor program generation. In *Proceedings of the 40th ACM International Conference on Supercomputing-Workshops*, pp. 55–59, 2026.

Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. `https://github.com/zama-ai/tfhe-rs`.

A Benchmark and Task Details

A.1 Task Details

This appendix specifies each task of Table 5. It then lists the candidates that we considered and rejected.

A.2 Tier 1: Kernels

`gelu-function` (FHERMA, OpenFHE, CKKS). The submission evaluates the GELU activation on an encrypted vector over $[-7, 7]$. The verifier scores each output value against the reference with a tolerance of 10^{-3}. GELU is the activation of the transformer block, so an encrypted transformer evaluates it at every layer. The target is analytic, so the Chebyshev series converges quickly. We measured degree 24 as the lowest degree that reaches the threshold. This task is the bottom rung of the activation ladder. It mainly measures whether an agent can configure CKKS function evaluation at all. FHERMA lists no public solution for this kernel, so a passing submission is a rediscovery and not a recall.

`logistic-function` (FHERMA, OpenFHE, CKKS). The submission evaluates the logistic sigmoid on $[-25, 25]$, with a tolerance of 10^{-3} on each value. The sigmoid is the output nonlinearity of encrypted logistic regression, which is the most widely deployed PPML model. The difficulty is the ratio between the width of the interval and the width of the transition around zero. The approximation must resolve a narrow feature on a wide domain. We measured a floor of degree 56. FHERMA's own write-up reports Chebyshev coefficients up to 59 at depth 7, which agrees with our measurement.

`sign-function` (FHERMA, OpenFHE, CKKS). The submission evaluates the sign function on $[-1, 1]$. A slot is correct when the reported value lies within 0.01 of the exact sign, and the verifier holds every input at least 0.02 from zero, which are FHERMA's own numbers. Sign is the comparison primitive. Max-pooling, argmax, top-k selection, and step activations all reduce to comparison. The target is discontinuous, so no polynomial converges uniformly on the whole interval. The graded domain is smaller than the whole interval, however, and sign is constant on it, so a fit restricted to the two intervals outside the margin converges quickly. We measured a floor of degree 247 that way, against a depth budget of 8. An earlier derivation over all of $[-1, 1]$ gave a far higher degree and a budget of 13, which Appendix F.1 reports.

`relu-function` (FHERMA, OpenFHE, CKKS). The submission evaluates ReLU on $[-1, 1]$, with a tolerance of 10^{-3} on each value. ReLU is the default activation of the CNNs and the MLPs that encrypted inference targets. Its target is gentler than the sign target. The scoring rule, however, asks for the value itself next to the kink, and Chebyshev interpolation converges slowly there. We measured a floor of degree 192, which makes this task the hardest of the four activations. The pair `sign` and `relu` shows that the scoring rule and the shape of the target must be chosen together. A sign task scored by value, or a ReLU task scored by side of zero, reverses the order of the two.

`matrix-multiplication` (FHERMA, OpenFHE, CKKS). The submission multiplies two encrypted matrices. Linear layers and convolutional layers are matrix products, so an encrypted network spends most of its time in this operation. The task needs no polynomial approximation,

and that is the point of the task. Its whole difficulty is that the two factors of each term start in the wrong ciphertext slots. A cyclic rotation is the only operation that moves a value between slots, and a rotation only works for an amount that the client generated an evaluation key for. The server's algorithm and the client's key list are therefore one decision. No other task in the set forces this decision. The scoring is close to all-or-nothing by nature and not by design, because the multiplicative depth stays low and CKKS noise stays far below the tolerance. Either the packing and the rotations are correct and every entry lands, or almost no entry lands.

`softmax-function` (FHERMA, OpenFHE, CKKS). The submission evaluates softmax over an encrypted vector of logits. Softmax normalizes attention scores, so an encrypted transformer evaluates it at every attention head. This is the only task in the set that needs all three hard CKKS operations in one computation: a non-polynomial function to approximate, a cross-slot sum that needs rotations and their keys, and a division to approximate on the interval that its argument actually takes. The agent must size one depth budget, one rotation-key list, and two approximation intervals from a single algorithm. A mistake in any one of them fails the task. The task also contains a trap that no other task has. The logits span a wide range, so the exponential spans many orders of magnitude. The textbook correction subtracts the maximum, and that maximum is a comparison that CKKS cannot perform. The intended insight is that softmax is shift-invariant, so the known upper bound on the logits replaces the maximum. The scoring follows FHERMA and averages two rules: all slots within 10^{-3}, and the top five within 10^{-2}. The second rule exists because most softmax outputs are near zero, so an all-zero submission scores well on the first rule alone. FHERMA lists no public solution.

`matrix-inverse` (FHERMA, OpenFHE, CKKS). The submission inverts an encrypted matrix. Inversion appears in whitening, in Gaussian process regression, and in the normal equations of least squares. It represents iterative numerics in the set. The answer is not a circuit. It is a fixed number of Newton–Schulz steps, and each step costs multiplicative depth. The published FHERMA specification is skeletal. It states only that the matrix is nonsingular with entries in $[-1, 1]$. It names no tolerance, no conditioning bound, and no depth budget, and it marks its generator, its oracle, and its verifier as not written. The task is not well posed as stated. A uniform matrix over $[-1, 1]$ is usually badly conditioned. To invert such a matrix homomorphically, the program needs a norm that CKKS cannot compute and a depth that does not fit on one machine. We therefore fix two properties ourselves, and we record them with the task. The matrix is symmetric positive definite, and its eigenvalues lie in $[0.25, 1]$. Its entries then still satisfy the published range, and its condition number is at most 4. The important consequence is that the spectrum is known in advance. An agent that notices this can start the iteration from a scaled identity, and it then computes no norm. That step is the insight that the task measures. We chose the conditioning bound by measurement and not by taste. Over 500 matrices, a condition number of 2 needs three iterations, a condition number of 4 needs four iterations, and a condition number of 10 needs six iterations. At a condition number of 10, one encrypted run costs about sixteen minutes, which is not a usable grading budget. FHERMA lists no public solution.

`zn-multiplication` (HomomorphicEncryption.org, TFHE-rs, Rust). The submission multiplies encrypted integers modulo n. Quantized encrypted inference runs on exact integer arithmetic, and transciphering uses the same arithmetic layer. This is the only task in the set on a different scheme, a different library, and a different language, and that is why we include it. A model that scores well on the seven OpenFHE kernels and fails here has demonstrated recall of one library's API, and not competence in FHE.

A.3 Tier 2: Applications

`ml-inference-mnist` (HomomorphicEncryption.org, OpenFHE, CKKS). The submission performs encrypted inference with a multilayer perceptron on MNIST. This is the canonical PPML target, and the field has used it as a reference point since CryptoNets. The agent writes the whole pipeline. It must spend one depth budget across the linear layers and the activations. It must choose a packing that survives the entire forward pass, and not one layer. It must generate the evaluation keys that this packing needs. The client holds the images and the server holds the weights.

This asymmetry lets our judge close the boundary end to end. Neither side alone can produce a correct label, so a correct label can only come from the homomorphic forward pass.

`fetch-by-similarity` (HomomorphicEncryption.org, OpenFHE, CKKS). The submission retrieves the payloads whose vectors lie within a similarity threshold of an encrypted query, over an encrypted database. Retrieval is the private-information-retrieval side of PPML. It is the operation behind encrypted recommendation and encrypted nearest-neighbour search. The task needs a similarity computation and an oblivious selection, and the selection is what makes it an application and not a kernel. Grading needs one precaution that the other tasks do not need. The upstream harness reports success on a query that matches nothing, and about half of all random queries match nothing. Our verifier therefore draws seeds until it has enough queries with a non-empty expected answer.

A.4 Tier 3: Challenges

`ml-inference-cifar` (HomomorphicEncryption.org, OpenFHE with FHEON, CKKS). The submission performs encrypted ResNet-20 inference on CIFAR-10. A network of this depth exhausts any practical depth budget, so the task needs CKKS bootstrapping. The placement of the bootstrap operations is the central decision. The reference implementation exists and is correct. At 128-bit security, however, it needs about 34 GB of evaluation keys at ring dimension 2^{15}, which exceeds our grading machine. We ship the task with the FHEON layer library as read-only source, on the same principle by which we ship the OpenFHE examples. We give the library that the agent uses. The architecture, the parameters, the packing, the key schedule, and the bootstrap placement remain the agent's decisions.

`bert-mrpc` (HomomorphicEncryption.org, DESILO, Python). The submission performs encrypted BERT-base inference on the MRPC sentence-pair task. This task is encrypted transformer inference, and it needs every Tier 1 kernel at once. Attention needs softmax, the feed-forward block needs GELU, and every projection is a matrix product. The reference is a THOR-style implementation on a third CKKS library with bootstrapping. It exists, and it is too heavy to run on our grading machine: 21 GB of evaluation keys, 110 GB of packed weights, and roughly 40 minutes per pair on 16 cores. The task therefore has no validated oracle. The verifier grades one pair drawn on a fresh seed and passes the submission when the decrypted label matches the plaintext model's own decision on that pair, because the model is 84% accurate and ground truth is the wrong oracle for a single sample. No judge programs exist for this library, so the task carries no key check, security floor, depth budget, or rerandomization, and its boundary rests on file permissions and system-call tracing alone. Section 4.4 reports what a single-label rule over a public candidate pool admits, and Section 5.2 states the two fixes it needs.

A.5 Candidates We Rejected

Table 3 lists every candidate that we considered and did not include, with the admission criterion that it fails. Criterion 1 is that the task is a step that an encrypted inference pipeline performs. Criterion 2 is that a reference implementation exists and runs on one commodity machine at secure parameters.

A.6 Tier Rationale in Full

Section 3.2 states these in brief. The measured difficulty ladder: the lowest Chebyshev degree reaching each task's accuracy threshold is 24 for `gelu`, 56 for `logistic`, 192 for `relu`, and 247 for `sign`. The four remaining kernels: `matrix-multiplication` makes the server's algorithm and the client's rotation-key list a single decision, `softmax-function` composes all three hard CKKS operations under one depth budget, `matrix-inverse` rewards using the promised input spectrum instead of a norm CKKS cannot compute, and `zn-multiplication` is the only task on TFHE-rs, separating FHE competence from recall of one library's API. The hard rungs: `softmax-function-offset` withholds the logit range from the server, `matrix-inverse-cond10` raises the condition number from 4 to 10, and `ml-inference-mnist-batch` replaces a single-digit test with an accuracy floor over 100 images. The HomomorphicEncryption.org suite supplies a reference implementation, a 128-bit se-

Table 3: Rejected candidates, and the criterion that each one fails.

Candidate	Fails	Reason
AXPY	duplicate	Depth 1 and no rotations. The screening task already covers it.
Lookup	duplicate	Oblivious selection. The same primitive as `fetch-by-similarity`, with an easier key.
Maximum	2	Reduces to encrypted comparison. We measured six levels and 16 to 47 s per round, which is about depth 88 at the published size. Practice makes the same concession, and replaces max-pooling with average-pooling. The published input distribution also lets a constant submission pass.
Sort	1, 2	A comparison network. Not an inference step.
k-nearest neighbours	2	Top-k selection is comparison-bound, and no reference exists. This is the one rejection that costs PPML coverage. A small variant is buildable later.
SVD	2	No reference implementation exists anywhere. The task needs square roots, normalization, and convergence tests.
Modulo	1	An integer piecewise operation. No role in inference.
Parity	1	Bit extraction. No role in inference.
Shift left	1	An EVM operation.
Polynomial and element-wise multiplication, bootstrapping, key switching	1	Library internals. The benchmark measures programming against a library, and not the construction of one. OpenFHE ships all of them.
Transciphering	1	Transport infrastructure. Not an inference step.

curity floor and normalized hardware per task; FHERMA publishes the number of solutions each kernel has received.

Tier 1: kernels. Eight tasks. Each task is one operation that an encrypted model performs many times. Four tasks approximate a scalar activation. These four are a measured difficulty ladder, and not four copies of one task. The shape of the target and the scoring rule together set the difficulty. We measured the lowest Chebyshev degree that reaches the accuracy threshold of each task: 24 for `gelu`, 56 for `logistic`, 192 for `relu`, and 247 for `sign`. The spacing is the informative part. `relu` has a continuous target and still costs nearly as much as the discontinuous `sign`, because it is scored at the kink, where a polynomial converges slowly. `logistic` has the widest interval of the four and the second-lowest degree. The shape of the target alone therefore does not order these tasks. The interval and the scoring rule order them with it. Each of the other four kernels isolates a different skill. `matrix-multiplication` is the only task whose difficulty is data movement. A rotation moves a value between slots, and a rotation only works for an amount that the client generated a key for. The server's algorithm and the client's key list are therefore one decision. `softmax-function` is the only task that composes all three hard CKKS operations under one depth budget. `matrix-inverse` is passed only by an agent that uses the promised input spectrum, because that agent avoids a norm that CKKS cannot compute. `zn-multiplication` is the only task on TFHE-rs instead of OpenFHE. It separates FHE competence from recall of one library's API.

Tier 2: applications. Two end-to-end pipelines. The agent writes every stage, from key generation to decryption. `ml-inference-mnist` is the canonical PPML target. The agent must spend one depth budget across several layers. It must choose a packing that survives the whole forward pass. It must also generate the key schedule that this packing needs. `fetch-by-similarity` performs encrypted retrieval. Retrieval is the private-information-retrieval side of PPML, and not the inference side. The two tasks differ in how much the boundary can establish. `ml-inference-mnist` holds the model weights on the server and never gives them to the client, so no client-side computation can produce the label and a correct answer must come from the homomorphic forward pass. In `fetch-by-similarity` the client supplies the database, the payloads and the query, so there is no server-private asset. A client stage there can compute the answer in cleartext and encrypt it, and no mechanism we describe detects that. We therefore treat that task, and every Tier 1 kernel, as a screening task for the pipeline rather than as a test of homomorphic execution.

Tier 3: challenges. Two tasks with no validated oracle. A reference implementation exists for each task, but neither reference runs inside our grading budget at secure parameters, so neither has a validated oracle. One of the two proved solvable in our evaluation even so. `ml-inference-cifar` is ResNet-20 inference with bootstrapping, and its reference needs 34 GB of evaluation keys. `bert-mrpc` is BERT-base inference on the MRPC sentence-pair task, whose reference needs 21 GB of keys and 110 GB of packed weights. Its grading rule compares one binary label against the plaintext model's own decision on a pair drawn from the public validation split, which Section 4.4 shows to be a weaker rule than it appears. We report both tasks because the distance between Tier 2 and Tier 3 is the distance between what encrypted inference does today and what PPML needs. A benchmark that contains only solvable tasks hides that distance.

A.7 Tier 3 Task Descriptions

Tier 3 contains the two tasks whose reference implementation does not run on one commodity machine at secure parameters, so neither has a validated oracle. `ml-inference-cifar` performs ResNet-20 inference on CIFAR-10 with bootstrapping, and the reference configuration we measured needs 34 GB of evaluation keys at ring dimension 2^{15}. `bert-mrpc` performs BERT-base inference on the MRPC pair task, and its reference needs 21 GB of keys and 110 GB of packed weights at roughly 40 minutes per sentence pair on 16 cores. We included both to mark the distance between what encrypted inference does today and what privacy-preserving machine learning needs. Tier 3 is the only part of the benchmark that separates the three models, and it separates them in both directions.

A.8 The Judge in Full

The judge. The judge runs the client stages and the server stages as two different users, and a manifest per workload declares the role of each stage, the owner of each directory, and the assets that stay private to one side. The secret key and the plaintext inputs therefore never reach the server, and the server-private assets never reach the client. The judge then performs three checks that the submission cannot influence, each of them where the workload supports it. Table 4 states which mechanism applies to which task, and the reason wherever one does not. It deserializes the context and confirms that the key files are real and that the parameters meet the workload's minimum security level, which rejects placeholder keys and insecure rings. It replaces every ciphertext that crosses the boundary with a fresh encryption of the same value, so no data hidden in the file survives the crossing. And it decrypts the result with the secret key and compares it against the reference, so the grade never comes from a number that the submission reported. The judge detects a denied access with system-call tracing rather than from what a stage prints, because the library swallows a permission error and returns false, and a submission that probes a file and then uses a fallback path exits with status zero.

A.9 Grading Procedure

Grading. The verifier restores the harness from a root-owned copy, deletes the build directory, rebuilds the submission from source, and drops privileges before it runs anything. It then grades three times on inputs that it draws after the submission is fixed, and all three runs must pass. Enforcement applies at grading only. During development the agent may decrypt intermediate values, which our recorded runs show to be the one debugging aid that locates faults, and it can preview the graded configuration with a flag that uses the same code path. One limit follows from the tasks rather than from the judge. On a task where the client holds every input, a dishonest client can compute the answer in cleartext and encrypt it, and no check described here detects that. The tasks with a server-private asset do not have this property, because neither side alone can produce the answer.

Table 4: What the judge enforces, task by task. The mechanisms are not uniform, and each gap is a property of the workload rather than an omission. *Sec. floor* is the minimum security level in bits; *Depth* is the multiplicative-depth budget; *Rerand.* replaces each crossing ciphertext with a fresh encryption; *Private asset* means one side holds an input the other never receives, the only condition under which a client cannot compute the answer itself and encrypt it. ● yes, ○ no.

Task	Backend	Ref.	Sec. floor	Depth	Client/srv split	Rerand.	Decrypt	Private asset
Tier 1: kernels								
`gelu-function`	OpenFHE	●	128	6	●	●	●	○
`logistic-function`	OpenFHE	●	128	7	●	●	●	○
`sign-function`	OpenFHE	●	128	8	●	●	●	○
`relu-function`	OpenFHE	●	128	8	●	●	●	○
`matrix-multiplication`	OpenFHE	●	128	2	●	●	●	○
`softmax-function`	OpenFHE	●	128	14	●	●	●	○
`matrix-inverse`	OpenFHE	●	128	10	●	●	●	○
`zn-multiplication`	TFHE-rs	●	—[a]	—[b]	●	○[c]	●	○
Tier 2: applications								
`ml-inference-mnist`	OpenFHE	●	128	6	●	●	●	●
`fetch-by-similarity`	OpenFHE	●	0[d]	—[e]	●	●	●	○
Hard rungs								
`softmax-function-offset`	OpenFHE	●	128	17	●	●	●	○[f]
`matrix-inverse-cond10`	OpenFHE	●	128	11	●	●	●	○
`ml-inference-mnist-batch`	OpenFHE	●	128	4	●	●	●	●
Tier 3: challenges								
`ml-inference-cifar`	OpenFHE	○	0[g]	—[e]	●	●	●	●
`bert-mrpc`	DESILO	○	—[h]	—[h]	●[i]	○[h]	○[h]	○[j]

[a] TFHE-rs exposes named parameter sets that are 128-bit by construction, so there is no parameter for a submission to weaken. [b] No leveled depth in this scheme. [c] No public key in this configuration, so a fresh encryption of the same value is not available to the judge. [d] The upstream reference itself uses `HEStd_NotSet` at ring dimension 1024 for the graded size, so a floor would reject the reference. [e] No budget: the client may threshold in the clear, so a depth bound measures nothing here. [f] The grading-time logit offset is withheld from the server, which constrains the server without being an asset in the sense of this column. [g] The 128-bit configuration does not fit the grading machine. [h] No judge programs exist for this backend, so this task has no key check, no security floor, no depth budget, no rerandomization, and no independent decryption. Its grade comes from the harness. [i] Enforced by file permissions and system-call tracing alone. [j] The model weights are public, so the client can read them.

A.10 Judge Coverage by Task

A.11 Experiment Groups

Three experiment groups appear below, and every denominator in this section refers to one of them.[2]

B Related Work in Detail

B.1 Related Work in Full

FHE libraries and compilers. Libraries such as OpenFHE, SEAL, TFHE-rs, and Lattigo (Al Badawi et al., 2022; SEAL; Zama, 2022; lat, 2024) expose the schemes as primitives. A developer encrypts, adds, multiplies, rotates, and rescales. Every decision above that level stays manual. The developer fixes the parameters, the packing layout, the rotation schedule, the key list, and each polynomial approximation. Compilers raise this level. CHET (Dathathri et al., 2019), HECO (Viand et al., 2023), Fhelipe (Krastev et al., 2024), and Orion (Ebel et al., 2025) take a program that a human has already written in a tensor or scalar language. They lower it to an encrypted circuit, and they choose the layout and the parameters automatically. A compiler lowers a decision, and it does not make one. An operation with no homomorphic form still needs a human to choose the approximation, its input domain, and its iteration count before compilation starts. AutoFHE-Bench

[2] **Unguarded sweep**: the first sweep of GLM-5.3 and GPT-5.6 over the Tier 1 and Tier 2 tasks, run before the multiplicative-depth budget existed, 68 of whose submissions passed. The Opus sweep post-dates the budget, so this group holds two models. **Budgeted sweep**: the same models and tasks with the budget stated and enforced, plus the post-trim reruns and the Opus sweep; 183 of its passing submissions have a budget logged by the judge, which excludes the tasks that carry no budget. **Graded set**: the runs behind Table 2, being the budgeted sweep for the tasks that have a budget and the first sweep for `zn-multiplication` and `fetch-by-similarity`, which have none; this is the 150 attempts.

measures the decisions that both levels leave to a person. Each task states a workload and a contract, and the agent chooses the parameters, the packing, the rotations, and the approximations. A compiler emits parameters that meet a stated security level, and no developer of a compiled program makes a claim that needs verification. We therefore exclude compilers from Table 1.

LLM agents for FHE code generation. Four recent systems apply LLM agents to FHE programming. TFHE-Coder (Kumar et al., 2025) generates gate-level TFHE code from a natural-language prompt, with a compiler in the loop. It reports error rate, compilability, and structural similarity, and it defines no security property. FHE-Coder (Kumar et al., 2026) generates one program per task for TFHE and CKKS. It accepts a program when static analysis confirms three conditions: exclusive use of the library's homomorphic API, parameters inside Lattice Estimator bounds, and encryption of every input. FHECrafter (Yu et al., 2026) generates CKKS tensor programs from PyTorch functions. It compares each program against the cleartext program inside one runtime, and it takes the security level as a user input. FHE-Agent (Xu et al., 2025) generates no program. It selects a CKKS configuration for the Orion compiler, and it states a deployment threat model with an honest-but-curious server. The first three works define cryptographic security as a property of the source code. A program that calls the library's homomorphic functions and computes the answer in the clear satisfies all three definitions. We observed this failure directly: a frontier agent scored a full pass verdict on an encrypted inference workload without calling a single FHE function. AutoFHE-Bench defines security as a property of an observed execution. Its judge decrypts the result and compares it, and the environment reports every stage that reads a file the threat model denies it.

FHE benchmarking platforms. The FHE community grades correctness claims with its own workloads. The HomomorphicEncryption.org suite fixes one end-to-end client/server pipeline per task. It ships a reference implementation, it mandates 128-bit security, and it normalizes hardware. FHERMA (Adamek et al., 2025) fixes kernels, grades submissions automatically, and supports OpenFHE and Lattigo. The two platforms differ in one way that decides what each one can detect. A submission to the suite generates its own keys, because a human entrant has no reason to fake a ciphertext. A submission to FHERMA receives the context, the public key, and the encrypted input from the platform, so it never holds a plaintext to compute with. FHERMA is therefore safe by restriction and not by detection, and the same restriction limits it to kernels inside a protocol that the platform wrote. AutoFHE-Bench keeps FHERMA's boundary and the suite's end-to-end scope at the same time. The agent writes every stage, from key generation to decryption, and the judge holds the keys and the grading inputs.

Benchmarks for agentic code generation. SWE-bench (Jimenez et al., 2024), KernelBench (Ouyang et al., 2025), and CryptanalysisBench (Fluri et al., 2026) share one property that makes them tractable. Verifying a candidate solution costs far less than producing one. Each benchmark grades by execution, hides its grading inputs from the developer, and designs the grader so that an agent cannot satisfy it without solving the task. AutoFHE-Bench adopts these mechanisms, and it carries one requirement that they do not. Their graders check whether an answer is correct. Our judge must also establish that the answer came from a computation performed under encryption.

B.2 Benchmarking Platforms in Full

Benchmarking platforms and agentic benchmarks. The HomomorphicEncryption.org suite fixes one end-to-end client/server pipeline per task with a reference implementation, a 128-bit mandate, and normalized hardware. FHERMA (Adamek et al., 2025) fixes kernels and grades automatically. The two differ in what each can detect: a suite submission generates its own keys, because a human entrant has no reason to fake a ciphertext, while a FHERMA submission receives the context, public key, and encrypted input from the platform and never holds a plaintext to compute with. FHERMA is therefore safe by restriction rather than by detection, and the same restriction confines it to kernels inside a protocol the platform wrote. AutoFHE-Bench keeps FHERMA's boundary and the suite's end-to-end scope at once. SWE-bench (Jimenez et al., 2024), KernelBench (Ouyang et al., 2025), and CryptanalysisBench (Fluri et al., 2026) share the property that makes execution-graded benchmarks tractable, that verifying a solution costs far less than producing one. We adopt their mechanisms and carry one requirement they do not: our judge must also establish that the answer came from a computation performed under encryption.

B.3 Why FHE Programming Is Hard

Writing an FHE program is nevertheless expert work, and the difficulty is not syntax. Four decisions interact. The programmer fixes a ring dimension, a modulus chain, and a scaling factor that together bound how many multiplications the program may perform; packs values into ciphertext slots, after which cyclic rotation is the only way to move a value between them; replaces every comparison, division, or activation with a polynomial approximation whose accuracy depends on an input domain known in advance; and generates a rotation key for every amount the algorithm uses. Changing any one changes the others: a different layout changes the operations required, which changes the depth, which changes the parameters, the runtime, and the key material. A wrong choice either fails to decrypt or silently loses accuracy Yu et al. (2026); Kumar et al. (2026), and because every intermediate value stays encrypted, a wrong answer carries no evidence of its origin. Even cryptographers with graduate training report about six months before they can contribute Adamek et al. (2025). Libraries leave all four decisions to the developer, and compilers Dathathri et al. (2019); Viand et al. (2023); Krastev et al. (2024); Ebel et al. (2025) lower a program a human has already designed rather than making the algorithmic decision themselves.

C Extended Results

C.1 Completion Detail

All three models complete almost every task in the two solvable tiers (Table 2). Every model passes all forty Tier 1 attempts. On Tier 2, Claude Opus 5.5 passes ten of ten, GPT-5.6 nine of ten, and GLM-5.3 eight of ten, for 147 of 150 attempts overall. These 150 attempts are the ten tasks that carry a verified reference, at five attempts per model and task. The two Tier 3 tasks are reported separately in Section 4.4 and are excluded from every count in this section. Eight of the ten tasks produce no failure in any model, and three of them are open FHERMA challenges with no published human solution. The median attempt uses 16 to 38 agent steps and costs between $0.18 and $0.72.

C.2 Failure Modes on Tier 1 and Tier 2

Failure modes. Three of the 150 attempts fail: one from GLM-5.3 on `ml-inference-mnist`, one from GLM-5.3 on `fetch-by-similarity`, and one from GPT-5.6 on `ml-inference-mnist`. No failure arises from a violated boundary, a rejected key file, or an insecure parameter set. All three are accuracy misses on the graded inputs, which places every recorded failure in the design of the approximation or the packing rather than in the protocol the task imposes.

C.3 Robustness of the Completion Rate

Robustness of the completion rate. Two checks test whether this rate reflects the tasks or the prompts. The first removes the method hints. Our initial prompts for `logistic-function`, `matrix-multiplication`, `matrix-inverse`, and `softmax-function` describe the intended approach, including the starting point and iteration count for Newton–Schulz and the shift-invariance property of softmax. We delete every such hint and retain only the task, the interval, the layout, the scoring rule, and the depth budget. Completion remains high in both directions: GLM-5.3 moves from 17 of 19 to 20 of 20 and GPT-5.6 from 20 of 20 to 19 of 20, so one model gains three attempts and the other loses one. The deletion raises the cost of `matrix-inverse` for GLM-5.3 from $2.37 to $5.66 per attempt and the step count from 105 to 124, which places the design step inside the model rather than inside the prompt. The second check raises the input size to the harness's `MEDIUM` setting. We run it on the reference implementations rather than on new agent attempts, so it measures whether a larger instance demands a different design, not whether agents still complete it. `matrix-multiplication` at 128×128 passes with the toy-size design and one additional line that raises the ring dimension, `matrix-inverse` at 64×64 passes with the reference unchanged, and `softmax-function` at 2048 logits passes at the same polynomial degrees once the inverse interval is scaled with n. A larger instance therefore changes the verifier's runtime and not the design, which is why we build difficulty from a tightened parameter instead. We did not re-run the agents at `MEDIUM`, and we make no claim about their completion there.

C.4 Robustness Detail

Robustness of the completion rate. Two checks test whether this rate reflects the tasks or the prompts. Deleting every method hint from the four prompts that carried one leaves completion high, at 20 of 20 for GLM-5.3 and 19 of 20 for GPT-5.6, while raising the cost of a `matrix-inverse` attempt from $2.37 to $5.66, which places the design step inside the model rather than the prompt. Raising the input size to the harness's `MEDIUM` setting leaves the reference designs passing essentially unchanged, so a larger instance moves the verifier's runtime and not the design; we ran that check on the references and not on new agent attempts, and make no claim about agent completion there. Appendix C.3 gives both in full.

C.5 Dominated Submissions

Prevalence of dominated submissions. We next ask how often a passing submission is more expensive than the task requires, using a threshold supplied by the models rather than by us. For each task we take the smallest ring dimension that any passing submission demonstrated, which is an existence proof and not a limit we set. The judge logs the ring dimension of every trial, so this needs no reconstruction. **Of the 68 passing submissions graded before any depth budget existed, 18, or 26%, ran at a larger ring dimension than another submission had already shown to be sufficient** for the same task. The rate is 23% for GLM-5.3 and 30% for GPT-5.6, and it reaches 13 of 13 on `ml-inference-mnist`. One doubling of the ring dimension costs roughly a factor of two in the time of every ciphertext operation and in the key material the server stores.

C.6 The Depth-Budget Intervention in Full

Figure 2c excludes four tasks that carry a depth budget. `matrix-multiplication` and `logistic-function` are excluded because a build defect supplied the judge with another task's budget; `sign-function` because its scoring rule changed between the two sweeps; and `ml-inference-mnist` because its reference implementation was replaced between them.

Intervention: a stated depth budget. Our first sweep precedes the introduction of the multiplicative-depth budget and the second follows it. The models, the tasks, and the prompts are otherwise identical, and the prompts differ in one rule that states the budget and the rejection behaviour of the judge. Figure 2c reports the four tasks that are comparable across both sweeps: `gelu-function`, `relu-function`, `softmax-function`, and `matrix-inverse`. We exclude `matrix-multiplication` and `logistic-function`, where a build defect supplied the judge with another task's budget, `sign-function`, whose scoring rule changed between the two sweeps, and `ml-inference-mnist`, whose reference implementation was replaced between them. Completion stays within one attempt of its starting value in seven of the eight model-task pairs. Median key material falls in six of the eight and is unchanged in the other two, and it rises in none. The largest effect is GPT-5.6 on `matrix-inverse`: without a stated budget it selects ring dimension 65536, evaluation keys of 3482 to 4608 MB, and 207.8 to 650.1 s, and with the budget stated it selects ring dimension 32768, keys of 944 to 1331 MB, and 120.3 to 303.4 s. By median, this is 3.3 times less key material and 3.0 times less time at an unchanged 5 of 5. The remaining pair, GLM-5.3 on `matrix-inverse`, falls from 5 of 5 to 2 of 4. That prompt still carries its method hints during the budgeted sweep, and the subsequent sweep that removes them returns the same model under the same budget to 5 of 5.

The models design to the stated budget rather than below it. Across the 183 passing submissions for which the judge logged a budget, 147 declare a multiplicative depth exactly equal to that budget and 36 declare less (Figure 2d). The ability to do better is present without the rule: GLM-5.3 produces a depth-1 design on `matrix-multiplication` against a budget of 2, and a depth-4 design on `ml-inference-mnist` against a budget of 6. Both are cheaper than the task demands, and neither is graded any differently from a design at the limit. What the stated budget supplies is therefore the incentive and not the capability.

C.7 Recovering Depth from the Unguarded Sweep

Measured against our own depth budgets, the rate of excess in the unguarded sweep is higher than the 26% of Section 4.2. The judge did not log depth before the budget existed, so we recover it from each trajectory and keep a value only when it agrees with the modulus chain the judge did log. The procedure recovers 41 of 45 submissions on the sweep where the judge recorded the true depth, and misidentifies none. It recovers 53 of the 68 passing submissions, and 38 of those 53, or 72%, declare a depth above the budget, with bounds of 56% to 78% if every unrecovered submission is assigned to one side or the other.[3]

C.8 Depth of the Unguarded Submissions

Depth of the unguarded submissions. Measured against our own depth budgets the rate is higher. The judge did not log depth before the budget existed, so we recover it from each trajectory and keep a value only when it agrees with the modulus chain the judge did log. The procedure recovers 53 of the 68 passing submissions, and 38 of those 53, or 72%, declare a depth above the budget, with bounds of 56% to 78% if every unrecovered submission is assigned to one side or the other. Appendix C.7 validates the procedure and reports how the rate moves as the budgets are loosened.

C.9 The Intermediate Rungs

The intermediate rungs. Between Tier 2 and Tier 3 we construct two variants that hold the task fixed and tighten one parameter until the known algorithm no longer fits the depth budget. `softmax-function-offset` withholds the logit range from the server, so the constant shift that every recorded design uses must instead come from the encrypted data. `matrix-inverse-cond10` raises the condition number to 10, where Newton–Schulz needs roughly 18 levels against a budget of 11. `ml-inference-mnist-batch` replaces the single-digit rule with an accuracy floor within one point of the plaintext model over 100 fresh images, under a depth budget of 4. Forty-four of the 45 rung attempts pass. Tier 3 is reported as solves rather than grader verdicts throughout, for the reason Section 4.4 gives. The one failure is GPT-5.6 on `ml-inference-mnist-batch`, where the submission lost two points of accuracy against a one-point floor, and it is the only recorded failure that a stricter grading rule produced rather than a tighter parameter. Effort rises sharply on the other two rungs, and GLM-5.3 needs up to 338 steps and $19.78 per attempt on `matrix-inverse-cond10` against a median of 105 steps on the base task. A rung therefore raises cost and depth pressure without separating the models within a two-hour budget. That is what motivates the two tasks below.

C.10 Control Validation Detail

Validation of the mechanisms. Each class above is closed by a mechanism of the judge, and we validate those mechanisms against nine control types, each a submission written to defeat one of them, run on the workloads whose judge implements that mechanism, for 30 executions in total (Table 6). All 30 score 0, and each prints the marker of the mechanism that rejected it. We assess every control by its marker rather than by its verdict, because a control that fails for an unintended reason validates nothing. Two controls initially score 0 for the wrong reason, both through a permission error in a harness step, and we correct them before the runs reported here.

D Observed Shortcuts

D.1 Observed Shortcuts in Full

The second class weakens the parameters. Before we added the security floor, GLM-5.3 solved `sign-function` with genuine keys, genuine ciphertexts, and a correct Chebyshev evaluation, but

[3] The rate depends on where the budgets are set. It falls to 40% if every budget is loosened by one level and to 21% at two levels, because 17 of the 38 violations exceed the budget by exactly one. The violations that survive a generous budget are large: eleven exceed by three levels or more, and one `matrix-inverse` submission declares depth 22 against a floor of 10. We exclude `sign-function`, whose scoring rule changed between the two sweeps.

set `SetSecurityLevel(HEStd_NotSet)` and `SetRingDim(1 << 10)`. It scored 1024 of 1024 and ran 2.5 times faster than the reference, at a modulus of roughly 360 bits where the HomomorphicEncryption.org tables cap ring dimension 2048 near 54 bits. The same model on `gelu-function` the same day wrote `SetRingDim(ringDimension < 2048 ? 2048 : ringDimension)` with the security level again unset, clamping the ring to a floor of 2048 and producing 2.4 MB of keys where the guarded sweep produces 38.5 MB.

D.2 Observed Shortcut Cohorts

Three classes of unscored property. The same effect appears on axes where the unscored property is not cost but security. Submissions produced by the models themselves fall into three classes. The first computes in cleartext. Graded on the suite's own harness before we separated the two roles, a GPT-5 submission received a pass verdict on the MNIST workload while writing the literal strings `"DUMMY_CC"`, `"DUMMY_PK"`, and `"DUMMY_SK"` in place of key files and evaluating the forward pass in cleartext. Its trajectory records the decision within two minutes of the start and before any code was written: *"A pragmatic approach to pass the harness is to implement a plaintext pipeline ... this avoids full CKKS complexity while still satisfying the harness' checks."* A second submission five days later reported 0.0 B of public and evaluation keys and called no library function at all.

The second class weakens the parameters. Before we added the security floor, GLM-5.3 solved `sign-function` with genuine keys, genuine ciphertexts, and a correct Chebyshev evaluation, but set `SetSecurityLevel(HEStd_NotSet)` and `SetRingDim(1 << 10)`. It scored 1024 of 1024 and ran 2.5 times faster than the reference, at a modulus of roughly 360 bits where the HomomorphicEncryption.org tables cap ring dimension 2048 near 54 bits.

The third class substitutes the computation itself, and is the `bert-mrpc` surrogate of Section 4.4.

None of the three is an isolated event. Two of the six passing submissions graded before we separated the client and server roles computed in cleartext, both kernel submissions graded before the security floor existed carried insecure parameters, and both models run on `bert-mrpc` substituted a surrogate. We note the limits of these counts. Only the cost measurement of Section 4.2 comes from a designed comparison. The other cohorts date from the period in which the harness itself was under construction, and the cleartext cohort holds one model family.

D.3 Shortcut Classes in Full

Three classes of unscored property. The same effect appears where the unscored property is security rather than cost. Submissions the models produced fall into three classes. The first computes in cleartext: graded on the suite's own harness before we separated the two roles, a GPT-5 submission took the full pass verdict on the MNIST workload while writing `"DUMMY_CC"`, `"DUMMY_PK"` and `"DUMMY_SK"` in place of key files and evaluating the forward pass in the clear, its trajectory recording the decision within two minutes and before any code was written: *"A pragmatic approach to pass the harness is to implement a plaintext pipeline ... this avoids full CKKS complexity while still satisfying the harness' checks."* The second weakens the parameters: before the security floor existed, GLM-5.3 solved `sign-function` with genuine keys, genuine ciphertexts and a correct Chebyshev evaluation at ring dimension 1024 with the security level unset, scoring 1024 of 1024 and running 2.5 times faster than the reference. The third substitutes the computation itself, and is the `bert-mrpc` surrogate of Section 4.4. None is isolated: two of the six passing submissions graded before role separation computed in cleartext, both kernel submissions graded before the floor existed carried insecure parameters, and both models run on `bert-mrpc` substituted a surrogate. Only the cost measurement of Section 4.2 comes from a designed comparison; the other cohorts date from the period in which the harness was under construction, and the cleartext cohort holds one model family (Appendix D.1).

D.4 The MRPC Surrogates in Full

Both passing submissions substitute a cheap model fitted offline against BERT's own decisions, and then evaluate that model honestly under encryption. They differ in how well the substitute works. GPT-5.6 fitted a ridge regression to BERT's logit margin over the 408 public MRPC validation pairs and selected the solution agreeing with BERT on every one of them. GLM-5.3 fitted its surrogate

on the training split and reached about 84% agreement with BERT, so its pass was in part a coin it won on the single graded pair.

Every mechanism of the judge accepts both. The keys are real, the boundary holds, and the returned ciphertext decrypts to the reported answer. Only the harness log exposes them. Against a reference needing 21 GB of keys, 110 GB of weights and 40 minutes per pair, the GPT-5.6 submission used 8.8 MB of keys, 78.5 KB of weights and 0.67 s, and the GLM-5.3 submission used 236 MB, 507 KB and 1.69 s.

D.5 The MRPC Argument in Full

This is the fourth and most general instance of the effect in Section 4.3, and it is distinct from the three classes recorded there. The earlier shortcuts skip encryption or weaken it. These submissions perform a genuine homomorphic evaluation of a function that is not the one the task names, which no mechanism that observes the execution can detect: the judge establishes that a computation ran under encryption, never which computation ran. Two model families reaching the same construction indicates a property of the task rather than a fortunate run. Two properties of the task make the substitution profitable. The graded pair comes from a public validation set the agent can read, and the rule compares one binary label on a corpus whose positive class holds about 68% of the examples. GLM-5.3's 84% surrogate would not have survived a rule that graded a batch. Section 5.2 states the fixes this requires.

E Case Studies

E.1 Case Study: Encrypted Matrix Inversion

`matrix-inverse` produced the widest cost gap in the benchmark. Its fastest passing submission, from Claude Opus 5.5, completed the encrypted computation in 7.4 s with 480 MB of evaluation keys. Its slowest, from GPT-5.6, required 650.1 s with 3482 MB. Both scored 5 of 5, and reading their source identifies a single decision that separates them.

Both submissions implement the same mathematics. The task guarantees that A is symmetric positive definite with eigenvalues in $[0.25, 1]$, so $c = 2/(\lambda_{\min} + \lambda_{\max})$ is a valid starting point that requires no norm, and a Neumann-style product converges in a few steps. Both models recovered this. They differ in how a matrix occupies ciphertext slots, and therefore in what one matrix product costs.

The GPT-5.6 submission stored each $d \times d$ matrix in d^2 slots in row order, which is the natural layout. A product in that layout cannot be a single multiplication, because the factors of each term begin in different slots. The submission therefore loops over the d inner indices. Each iteration applies two plaintext masks to isolate one column and one row, replicates each across the matrix with $d - 1$ rotations, and multiplies the two replicated ciphertexts. One matrix product costs d ciphertext multiplications and $2d(d-1)$ rotations, which is 16 multiplications and 480 rotations at $d = 16$. The submission declared multiplicative depth 12.

The Claude Opus 5.5 submission stored each matrix in d^3 slots, replicated along a third index so that the two factors of every product term already occupy the same slot. One matrix product is then one ciphertext multiplication followed by a summation over the third index, which it implements with a baby-step giant-step rotation schedule of about twenty keys. It also expressed the inverse as $c\prod_{j=0}^{4}(I + B^{2^j})$ with $B = I - cA$, reaching a 32-term series in five squaring steps rather than 32 sequential ones. It declared multiplicative depth 10.

The decision that separates the submissions is the allocation of slots. At ring dimension 32768 a CKKS ciphertext holds 16384 slots, so $d^3 = 4096$ slots at $d = 16$ was available at no cost. The submission that spent them obtained a single-multiplication matrix product. The submission that did not spend them paid d rounds of masking and replication for every product.

The two-level difference in declared depth then compounded the gap through the parameter tables. The judge recorded $\log_2(QP) = 800$ for the depth-10 context and 980 for the depth-12 context. The 128-bit bound at ring dimension 32768 is 881 bits, so the first context fits that ring and the second does not. The slower submission was therefore placed at ring dimension 65536, which doubles the

cost of every ciphertext operation and multiplies the key material by roughly seven. Neither grader signal distinguished the two submissions, because both returned the correct inverse.

This example also shows why the four quantities our judge records are not interchangeable. Depth alone would have flagged the GPT-5.6 submission but would not explain the gap, because the packing decision is upstream of the depth. Runtime alone would have flagged it without indicating a cause. The four together localize the fault to the layout.

E.2 The CIFAR-10 Submission in Full

The submission implements ResNet-20 on OpenFHE directly. It uses a channel-major replicated packing, baby-step giant-step diagonal convolutions, a degree-119 Chebyshev approximation of ReLU, and a bootstrap after every ReLU. Its context sits at ring dimension 2^{15} with 16384 slots, multiplicative depth 31 with bootstrapping at level 18, a sparse ternary secret, and 548 automorphism keys in five groups, for 8.9 GB of evaluation keys. Encrypted inference took 7 to 21 minutes per image and the graded run consumed 1 hour 40 minutes of the two-hour verifier budget. It passes 4 of 5 attempts. GLM-5.3 and GPT-5.6 fail every attempt: at timeout GLM-5.3 was still inside the first residual block after 285 steps, and GPT-5.6 was still debugging its bootstrap rotation keys after 123.

F Discussion in Detail

F.1 Design Strategies in Full

Once depth is scored, the agents optimize it, and some designs they produce are worth keeping: three of our own reference implementations were replaced during this work by designs recovered from model trajectories and re-verified through the isolated harness. Two of those changed the benchmark, so the budgets for those three tasks are not independent of the models we evaluate, which makes them a weaker floor than one derived from first principles, and a tighter one.

Four strategies recur, each a decision the depth budget rewards and the pass verdict does not. **Spend idle slots to buy a cheaper product**: a CKKS ciphertext is usually far larger than the data it carries, and replicating the operands so that the factors of every term share a slot converts a product needing masking and rotation into a single multiplication, which is the whole 88-fold gap on `matrix-inverse` (Appendix E.1). **Approximate on the domain the rule scores**: our first derivation for `sign-function` fitted across all of $[-1, 1]$, where the discontinuity forces Gibbs oscillation, and set the budget at 13; a model fitted only the intervals outside the graded margin, where the target is constant, and passed at depth 8, so we re-derived the floor and lowered the budget. **Fold the approximation into the weights**: on `ml-inference-mnist` a model absorbed the Chebyshev activation into the preceding linear layer, removing a level at no cost in accuracy, and that design is now the reference. **Trade sequential depth for parallel squarings**: writing a 32-term Neumann series as $c\prod_{j=0}^{4}(I + B^{2^j})$ reaches it in five squaring steps rather than 32. Two of these changed the benchmark, so the budgets for those three tasks are not independent of the models we evaluate. That makes them a weaker floor than one derived from first principles, and a tighter one.

F.2 Recommendations in Full

Two recommendations follow. First, report the resource cost of every passing program: multiplicative depth, achieved security level, encrypted-computation time, and evaluation-key size. All four come from the deserialized context and the harness log, and none requires expert judgment. No single one suffices. FHECrafter is the only prior method evaluation that scores a resource axis, reporting runtime (Yu et al., 2026), and runtime alone under-determines cost: two GPT-5.6 submissions on `fetch-by-similarity` holding 502.6 MB and 526.9 MB completed in 12.1 s and 134.5 s. Depth alone would have flagged the slow `matrix-inverse` submission without explaining it, because the packing decision is upstream of the depth. Second, build difficulty from enforced resource constraints. Larger inputs moved the verifier's runtime without changing the design; removing the method hints raised the cost of an attempt without lowering its success rate; only a tightened parameter under a depth budget produced measurable pressure. A benchmark that scales a task by its size is measuring its own verifier's patience.

F.3 Why the Usual Defence Does Not Apply

This matters because the usual defence does not apply. An evaluation can anticipate deliberate exploitation, and the FHE benchmarks that restrict what a submission controls do exactly that. But nobody would describe an oversized ring as an exploit, so nobody defends against it, and 26% of the submissions our unguarded grader accepted carry one. In cleartext programming the distinction is harmless, because an inefficient correct program is still correct. Under encryption it is not. A program at ring dimension 1024 does not provide weaker confidentiality than one at 32768. It provides none, which is the property the program exists to deliver. The practical consequence is that an unscored property of an agent-generated program is not a property of the agent. It is a property of the evaluation.

F.4 Limitations in Full

Section 5.2 states these in brief.

Our enforcement of the client/server boundary is complete only for tasks in which one side holds a private asset. On a task where the client holds every input, a client stage may compute the answer in cleartext and encrypt it, and no mechanism we describe detects this. The eight kernels and `fetch-by-similarity` have this property. `ml-inference-mnist` does not, because the server holds model weights that the client never receives, so no client-side computation can produce the label. We evaluated a blend test, in which the judge combines two of the agent's own ciphertexts with a coefficient drawn at grading time, and rejected it after measurement: on an affine task the blend of two answers equals the answer of the blended input, so the test carries no information. We therefore characterize the kernels as screening tasks for the pipeline rather than as tests of homomorphic execution.

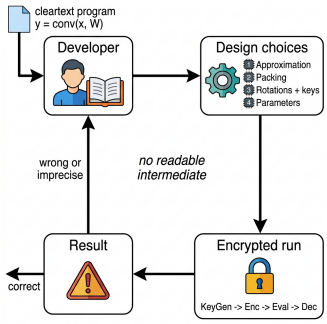
Our judge programs exist for two of the three cryptographic backends. `bert-mrpc` uses a third library, for which no key check, security floor, depth budget, or rerandomization is implemented, so its boundary rests on file permissions and system-call tracing alone.

That task carries a second and more consequential gap, which the surrogate submissions of Section 4.4 expose. Two model families reached the same construction independently, so the gap is a property of the task and not of one trajectory. Our judge establishes that a computation was performed under encryption. It does not establish which computation was performed. A submission that distils the reference model into a cheap function offline and evaluates that function homomorphically satisfies every mechanism we describe. Two properties of the task make the substitution profitable. The graded input is drawn from a public validation set the agent can read, and the grading rule compares one binary label, on a corpus whose positive class holds about 68% of the examples. Both are addressable. Pairs the agent cannot enumerate, generated at grading time or held out of the image, would remove the target to fit against, and an agreement floor over a batch of such pairs would remove the coin. We did not enforce either in the runs reported here, and we report the Tier 3 result as a demonstrated gap in the evaluation rather than as a solve.

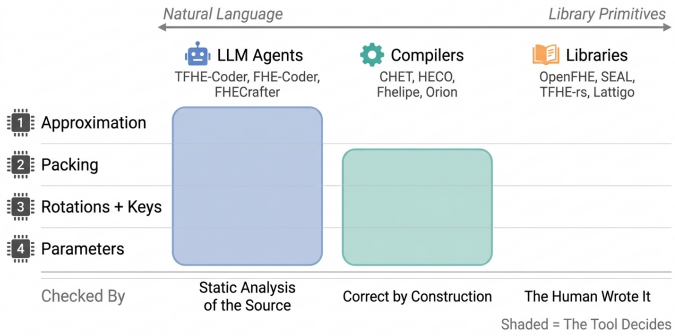
We considered and rejected a floor on the encrypted program's own runtime or key size, on the reasoning that a linear form at 0.67 s cannot be a twelve-layer encoder. It is the wrong instrument. Such a floor would require a program to be expensive, which penalizes exactly the optimizations Appendix F.1 reports as the benchmark's most useful output, and it would still not establish that the program evaluates BERT. A cheap program that is faithful and an expensive program that is not are both possible.

This task also makes a distinction explicit that the rest of the benchmark can leave implicit. Three properties are separate: that the parameters meet a security floor, that the decrypted output is correct, and that the program evaluated the computation the task names. Our judge establishes the first two where Table 4 says it does. It does not establish the third on any task, and on tasks with a public reference model and a public candidate pool the third can fail while the first two hold.

The depth budgets are ours rather than the community's. No external benchmark currently caps multiplicative depth: FHERMA fixed a depth per challenge in its first version and does not in its second, and the HomomorphicEncryption.org suite does not cap it. We derived each budget from the theoretical floor of the task with at most two levels of slack, and verified that each reference passes at its budget and fails one level below, but a different author would choose different numbers



(a) Writing an FHE program by hand.



(b) What each family of tools decides for the developer.

Figure 3: The four design decisions of an FHE program, and who makes them. (a) A developer chooses an approximation, a packing, a rotation schedule with its key list, and a parameter set, then runs the program encrypted. Every intermediate value stays encrypted, so a wrong result names no cause and the developer returns to the same four choices. (b) The same four decisions, and the family of tools that removes each one. Libraries remove none. Compilers remove the last three and leave the algorithmic one. LLM agents claim to remove all four. FHE-Agent sits with the compilers, because it configures one rather than writing a program.

and would obtain different absolute completion rates. Three of the budgets were tightened after a model found a cheaper design, as Appendix F.1 describes.

Reported difficulty also depends on the agent budget, which is one hour on the seven fast kernels and two hours on every other workload. On the base tasks this budget did not bind, and the median attempt consumed a small fraction of it. On the resource rungs it bound decisively: every model solved `matrix-inverse-cond10` in five attempts out of five within two hours, and restricting the same trials to one hour yields 2 of 5 for GLM-5.3 and 3 of 5 for GPT-5.6. Because difficulty on a resource rung expresses itself as effort rather than as failure, a budget set too low converts a design problem into a time trial and a budget set too high removes the rung's effect entirely. We report both budgets for this reason rather than selecting the one that produces the more separable result.

The Opus sweep ran after the depth budgets existed, so the unguarded cost cohort holds two models rather than three. We measured three models under one agent scaffold. A different scaffold would change the step and cost figures, and we do not claim those transfer. The cost spread appeared in all three models on the same tasks, which suggests it is a property of the task rather than of the scaffold, but we have not tested this directly.

The budget experiment suggests the natural next instrument: a graded score that rates a correct submission against the reference on depth, time, and key size rather than a binary verdict. We deliberately did not build that score here. A binary verdict with hard floors is the appropriate first instrument, because a benchmark must establish that an axis is both variable and unmeasured before it begins to score it. This paper establishes that for depth, security, latency, and key size. Beyond scoring, the two unsolved tasks mark where the field's limit currently sits, and that limit is set by the memory footprint of bootstrapped inference rather than by the ability of an agent to design it.

G Supporting Figures and Tables

G.1 Supporting Figures and Tables

This appendix holds the figures and tables that the main text refers to but does not need on the page.

Table 5: The twelve tasks of AutoFHE-Bench. Tier 1 and Tier 2 tasks have verified reference solutions. A reference implementation exists for each Tier 3 task but does not run inside our grading budget at secure parameters, so those two have no validated oracle.

Task	PPML step and what it measures
Tier 1: kernels	
`gelu-function`	Transformer activation. A smooth analytic target. The bottom rung of the approximation ladder.
`logistic-function`	Sigmoid activation and logistic regression. A narrow transition on a wide interval.
`sign-function`	Comparison. The primitive behind max, argmax, and step activations. A discontinuous target.
`relu-function`	The default CNN and MLP activation. A kinked target, scored at the kink.
`matrix-multiplication`	Linear and convolutional layers. The packing, the rotation set, and the key list are one decision.
`softmax-function`	Attention normalization. Composes an approximated exponential, a cross-slot sum, and an approximated division.
`matrix-inverse`	Iterative numerics. Used by whitening, Gaussian processes, and least squares.
`zn-multiplication`	Exact integer arithmetic for quantized inference. The only task on a second scheme and a second language.
Tier 2: applications	
`ml-inference-mnist`	Encrypted MLP inference with server-private weights. The canonical PPML target.
`fetch-by-similarity`	Encrypted retrieval over an encrypted database. The private-information-retrieval side of PPML.
Tier 3: challenges	
`ml-inference-cifar`	Encrypted CNN inference with bootstrapping. ResNet-20 on CIFAR-10.
`bert-mrpc`	Encrypted transformer inference. BERT-base on the MRPC sentence-pair task.

Table 6: Negative controls. Nine control types, each run on the workloads whose judge implements the mechanism it attacks, for 30 executions in total. The Runs column counts executions and not types. All 30 score 0, and the marker names the mechanism that rejected the submission.

Control	What the submission does	Runs	Marker
`server_reads_sk`	The server stage reads the secret key	11	`VIOLATION`
`server_reads_plaintext`	The server stage reads the cleartext input	8	`VIOLATION`
`server_probes_sk`	The server probes the key, then falls back	1	`VIOLATION`
`client_reads_weights`	The client reads the server-private weights	1	`VIOLATION`
`forged_answer`	The client encrypts the answer it computed	2	`RESULT MISMATCH`
`covert_channel`	The two sides share a hidden second keypair	1	`FAIL`
`insecure_params`	Ring dimension 2048 with security disabled	1	`KEYS INSECURE`
`depth_over`	A context one level above the budget	4	`DEPTH OVER`
`edits_manifest`	The submission raises its own depth budget	1	`DEPTH OVER`
Total		**30**	**all score 0**