Agentic Design Exploration for FPGA NTT Accelerators with Staged Tool Feedback

Nuo Xu[1], Pengju Liu[1], Xiaolin Xu[2], Wujie Wen[3], Zhenman Fang[1], and Caiwen Ding[1]
[1]University of Minnesota, Minneapolis, MN, USA
[2]Northeastern University, Boston, MA, USA
[3]North Carolina State University, Raleigh, NC, USA

***Abstract*—Parameterized generators automate number-theoretic transform (NTT) implementations on field-programmable gate arrays (FPGAs). Selecting which configuration to evaluate or revise next remains difficult because analytical estimates, synthesis results, and board measurements expose different constraints and performance tradeoffs. We present an agentic design-exploration framework that uses staged tool feedback to guide generator-constrained NTT configuration decisions. An agent based on a large language model (LLM) uses exploration history and returned observations to decide which supported configuration to evaluate, revise, or promote next. Deterministic validation checks each action, while generator components construct executable implementations when needed. Under the same three-request budget, the framework reduces the average percentage gap to the best feasible compute-time estimate in each candidate set compared with two predefined numerical rules. It also achieves synthesis feasibility by adjusting a configuration after a reported resource-limit violation. U280 measurements reveal resource and invocation-rate tradeoffs that intermediate timing estimates miss. Integration with two generator backends shows that the same agentic exploration loop supports different implementation flows.**

***Index Terms*—FPGA, number-theoretic transform, agentic design-space exploration, large language model, high-level synthesis**

I. Introduction

Designing a field-programmable gate array (FPGA) implementation of the number-theoretic transform (NTT) is a sequence of decisions made under partial information. Analytical models can screen many configurations cheaply, while functional simulation, high-level synthesis (HLS), physical implementation, and board measurements provide increasingly implementation-specific evidence at higher cost. Exploration therefore requires choosing which supported configuration to evaluate or revise next using the available stage-specific evidence.

NTT accelerators combine modular arithmetic with repeated data rearrangement for polynomial computation in lattice-based cryptography and fully homomorphic encryption. Architecture family, arithmetic parallelism, modular reduction, and memory organization jointly affect resource use and execution behavior [1], [2]. Designers therefore face coupled implementation choices under resource and performance constraints.

Parameterized generators make structured NTT design spaces executable. We reuse selected AutoNTT components for dependent-parameter derivation and implementation construction [1]. Our focus is the complementary decision problem: choosing which supported configuration to evaluate or revise next as stage-specific evidence becomes available. The generator remains responsible for construction, while the proposed framework maintains exploration state and makes evidence-conditioned next-action decisions. We also instantiate the same control/state flow with NGen as a second generator backend [3].

We introduce an agentic design-exploration framework that maintains an evolving Exploration State X_t over the workload, objective, available design alternatives, exploration history, tool observations, and remaining evaluation budget. A single Design Exploration Agent, guided by a large language model (LLM), uses this state to select an existing candidate, propose a supported configuration, or revise one already under consideration. Deterministic validation and budget checks cover legality, configuration identity, stage compatibility, and available budget before the system accepts a construction or evaluation request. For accepted construction requests, reused AutoNTT components derive dependent parameters and construct the implementation. Deterministic validation checks legality, and execution tools supply the metrics.

The framework integrates analytical estimates, functional simulation, HLS, physical implementation, and board measurement as distinct sources of stage-specific evidence. The framework requests higher-cost evaluations selectively, and completed stage observations can update the Exploration State for subsequent decisions. These returned observations form the staged tool feedback used by the exploration loop. The framework can identify candidates for physical implementation and board measurement; operators schedule and execute these stages.

We evaluate the framework through catalog replay (recorded synthesis results for fixed candidate sets), fresh generator-backed proposals, and operator-scheduled U280 studies. Numerical rules provide non-agent comparisons; feedback/no-feedback is an internal ablation of the same agent.

In catalog replay, the agent achieves a 3.223% average gap to each candidate set's best feasible compute-time estimate under three requests, below both predefined numerical rules. Its no-feedback ablation yields 5.154% across 36 paired episodes. Returned observations redirect exploration toward HLS-feasible candidates deprioritized by analytical

Accepted to special session track.

caps. An HLS resource violation also guides the agent to adjust parameters and obtain a feasible implementation; U280 measurements reveal resource and invocation-rate tradeoffs beyond intermediate timing. Across four 40-bit fresh settings, the configurations selected by AutoNTT's analytical search exceed the HLS resource gate of 260,000 lookup tables (LUTs), with HLS-to-analytical LUT ratios of 1.52–1.82×. This disagreement gives a concrete reason to reconsider a configuration after tool evaluation instead of committing to its analytical ranking.

Within each controlled study, we hold the objective, action domain, construction path where applicable, evaluation criteria, and request budget fixed across the decision methods in each comparison. These controls isolate configuration-selection quality; U280 measurements characterize later-stage hardware tradeoffs.

The contributions of this work are:

- A *stateful agentic design-exploration loop* that maintains an evolving Exploration State and uses one Design Exploration Agent for sequential configuration decisions, with deterministic action validation, explicit budget control, and generator-backed construction when needed.
- *Selective integration of stage-specific evidence* from analytical estimation, functional simulation, HLS, physical implementation, and board measurement, with the Design Exploration Agent using completed observations to condition subsequent configuration decisions. Operators selectively schedule higher-cost physical evaluations.
- *Evidence of improved configuration selection and constraint-aware adjustment*: lower average gaps than predefined numerical rules in three-request catalog replay, feedback-guided recovery of HLS resource feasibility, and resource–invocation-rate tradeoffs from operator-scheduled measurements.

Section II reviews background and related work; Section III presents the framework; Section IV reports the evaluation; Section V discusses limitations; and Section VI concludes.

II. Background and Related Work

A. Parameterized NTT Design and Generation

For transform length N, the radix-2 implementations considered here use $\log_2 N$ butterfly stages with repeated data rearrangement [1]. FPGA implementations trade butterfly parallelism against resource reuse and memory organization, while the modular-reduction method changes arithmetic cost [1], [2]. Parameterized generators expose these choices and construct the corresponding hardware.

AutoNTT provides iterative, dataflow, and hybrid architecture families with tunable parallelism and reuse [1]. It also supports multiple modular-reduction implementations and derives dependent parameters before generating HLS designs. In this work, AutoNTT provides the supported configurations and components used for dependent-parameter derivation and implementation construction.

Other NTT generators expose different design choices. OpenNTT automates memory access and twiddle generation [2]; NTTGen generates Verilog across data, pipeline, and batch parallelism [4]; and Proteus provides parameterized pipelined NTT generation [5]. These generators expose structured NTT design spaces with executable implementations.

B. Related Work on Hardware Design-Space Exploration

HLS design-space exploration (DSE) uses heuristic or numerical search over directives and implementation parameters [6]. AutoDSE applies bottleneck-guided coordinate optimization [7], while FADO jointly considers HLS directives and floorplanning on multi-die FPGAs [8]. These methods use tool-reported timing and resource results to guide search. Multi-fidelity methods additionally combine inexpensive early-stage estimates with costly later-stage results, modeling the earlier stages as correlated approximations of the final objective within a statistical surrogate [9], [10].

Recent LLM-based systems explore hardware designs at different levels. LLM-DSE coordinates routing, parameter proposal, selection, and critique in a feedback-guided search over HLS directives [11]. iDSE combines design-space pruning and representative initial sampling with HLS-result-guided refinement toward a Pareto front [12]. SECDA-DSE generates template-constrained SystemC accelerators and incorporates simulation, synthesis, and FPGA execution in its refinement workflow [13]. These systems already use returned tool results to guide later design choices. TimelyHLS, Agentic-HLS, and ChatEDA instead emphasize HLS code refinement, metric prediction, or design-tool coordination [14]–[16].

In the main studies, the decision object is an AutoNTT-supported NTT configuration, including architecture family, independent parallelism parameters, and registered implementation changes. The agent selects actions; deterministic validation and generator components determine whether a proposal can become an executable candidate. Our focus is how stage-specific evidence changes subsequent choices under a bounded evaluation budget. We retain analytical resource scores, HLS feasibility and compute-time estimates, and board invocation rates as distinct observations with different resource scopes and objectives (Section IV-B). Matched feedback/no-feedback and numerical controls test decision quality, while physical studies examine later-stage hardware tradeoffs.

III. Agentic Design Exploration Framework

The framework defines a sequential exploration loop for generator-backed NTT design (Fig. 1). A single LLM-guided Design Exploration Agent chooses a supported action from an evolving state. Deterministic validation and budget checks precede execution; returned observations inform later rounds.

The design separates next-experiment selection from candidate construction and measurement. A task specifies a workload, an objective, resource constraints, a supported configuration space, and a request budget. Analytical estimates provide an initial basis for selection; later tool results expose additional

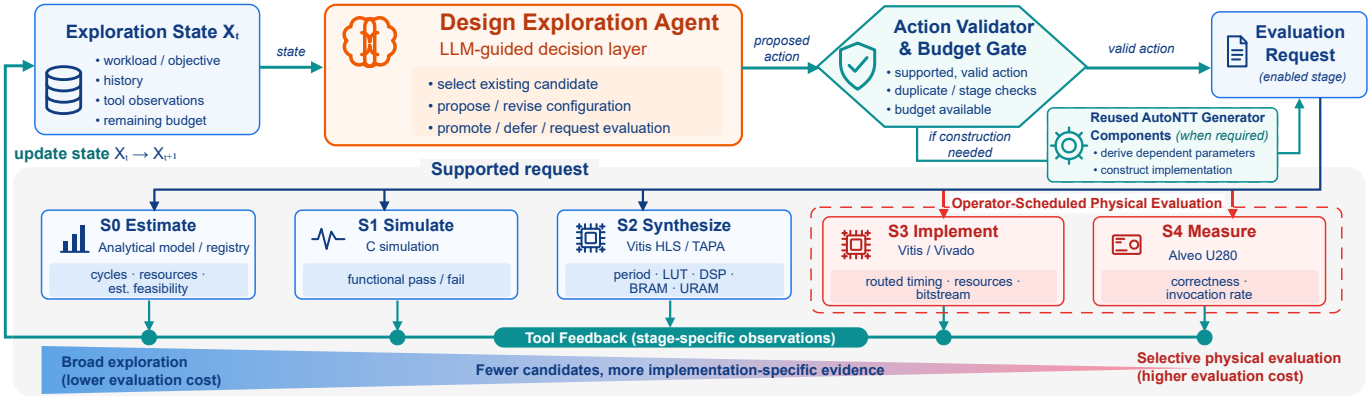


Fig. 1. AutoNTT exploration with staged tool feedback. The evolving state X_t conditions one LLM-guided Design Exploration Agent. Deterministic validation and budget checks constrain each supported action. Selected AutoNTT components construct an implementation only when a fresh proposal requires it. Operators schedule physical implementation and board measurement; returned records can enter a later decision round.

constraints and tradeoffs. Task-relevant evidence guides the loop toward a feasible configuration within budget.

A. Decision Context and Supported Actions

Each decision round t uses an evolving Exploration State, defined in Eq. (1), as

$$X_t = (W, O, \mathcal{C}_t, H_t, E_t, B_t)\,, \tag{1}$$

where W is the workload, O is the exploration objective and exposed constraints, $\mathcal{C}_t$ is the design context available to the current decision, H_t is the exploration and action history, E_t is the stage-specific evidence visible to the agent, and B_t is the remaining evaluation budget state. The candidate context $\mathcal{C}_t$ includes existing candidates and eligible proposal parents. Registration precedes evaluation: a constructed candidate can exist before tools establish its stage success.

For existing candidates, the context supplies configurations, analytical records, and study-visible observations. For fresh proposals, it also supplies eligible parents and their registered parameter changes. An unevaluated candidate or resource-infeasible parent remains available for supported revision. Stored results enter the next decision context according to the study's visibility rule; no-feedback retains candidate and action history while excluding returned tool observations.

The decision prompt asks for the best feasible observed outcome within the remaining requests. Resource limits determine which observations qualify, and the objective orders qualifying outcomes. An estimate guides a request, while a completed evaluation establishes the outcome used for comparison. The context therefore identifies previously requested candidates separately from their returned metrics. When a resource observation is visible, the agent can distinguish an untested alternative from a known constraint violation.

The Design Exploration Agent performs the model-guided decision step. We write its proposed action abstractly as $a_t = \pi(X_t)$, where π denotes the agent's abstract action-selection mapping and the available action subset depends on the current study. Supported actions may select or reuse an existing candidate, propose or revise a supported configuration from an eligible parent, promote an eligible candidate toward an enabled evaluation, or defer further evaluation or stop where supported. The enabled execution path determines the evaluation stages available to each model-selected action.

Catalog selection requests the saved outcome of an existing configuration. Fresh revision changes an eligible parent's independent parameters to test a new implementation. The prompt exposes allowed changes and the parent's analytical and visible stage records. The agent returns a candidate identifier or a parent, registered change, and independent parameter values, together with an evidence-based reason. The study fixes the objective and action domain; the agent chooses within that domain as observations arrive.

B. Deterministic Validation and Generator Construction

Deterministic validation and budget checks ground each proposed action before the system accepts a candidate resolution, construction, or evaluation. These checks cover the supported action schema, candidate identity, registered configuration change, parameter legality, stage prerequisites and compatibility, duplicate or equivalent requests, available budget, and study-specific objective or approval constraints where required. We denote this grounding abstractly as $\hat{a}_t = V(a_t, X_t)$, where $\hat{a}_t$ is the accepted action when validation succeeds. A rejected request creates no new implementation or valid stage result; the evaluation protocol in Section IV-A defines exact request charging.

The construction path depends on the accepted action. For AutoNTT catalog replay, the system resolves the registered candidate and attaches evidence exposed by the current study without reconstructing the implementation. For a fresh proposal or revision, the action identifies an eligible parent, a registered configuration change, and the independent parameters to vary. After validation, selected AutoNTT generator components derive dependent parameters and materialize the supported implementation for the enabled simulation/HLS evaluation path [1]. AutoNTT therefore serves as the deterministic construction boundary for fresh proposals. The framework registers a materialized candidate

before evaluation and records stage success or failure only after the requested evaluation returns.

The framework's execution adapters translate the accepted request into generator and tool invocations. In fresh AutoNTT evaluation, the validated configuration is passed to code generation and the enabled driver invokes simulation and HLS; the returned status and metrics become candidate-associated observations for the next decision context. The agent supplies the configuration request and reason, while the adapter supplies the executable tool calls.

The supported AutoNTT configuration schema defines the proposal space. The agent selects independent parameters and registered configuration changes exposed by the chosen architecture family. The generator derives dependent parameters; deterministic validation enforces family-specific legality. For example, under the supported hybrid mapping, with vertical size 8 and concatenation 1, the generator derives a horizontal size of 4, yielding $8 \times 4 \times 1 = 32$ butterfly units.

The framework identifies a candidate by the workload and implementation-defining configuration parameters. Equivalent aliases share one identity. For fresh construction, the validator rejects a canonical signature already present in the exploration state's known configurations, including implementation variants. For example, equal butterfly-unit counts do not identify the same implementation when a registered buffer variant changes its task organization. Including such variants in the signature distinguishes a new implementation experiment from a request for an existing configuration.

NGen [3] reuses the proposal, validation, evaluation, and state-update flow with backend-specific checks and tools (Section IV-D). AutoNTT actions vary supported architecture parameters and construct HLS implementations; NGen actions select registered lane, processing-element, and reduction configurations and regenerate register-transfer-level (RTL) implementations. AutoNTT returns HLS period and FPGA resource estimates, while NGen returns simulation cycles and generic synthesis cells. Each backend supplies its own context, action checks, construction, tools, and metric interpretation; the task fixes its objective and feasibility test. The common control flow therefore preserves backend-specific parameter relations and metrics.

C. Stage-Specific Evidence for Agent Decisions

The AutoNTT backend distinguishes five stage-specific evidence sources at different costs and levels of implementation specificity. They answer different design questions, not progressively more accurate versions of one scalar objective. Stage identity and returned metrics determine what the next supported decision should resolve. In the reported replay and fresh studies, the agent chooses the configuration; the driver fixes the saved-outcome query or simulation/HLS sequence. S0 denotes analytical estimation or an available analytical record. The agent compares estimated resource use and compute cycles of supported choices before spending a tool request. The initial context also exposes the independent parameters that produce those estimates. A later discrepancy can therefore lead either to testing another catalog entry or to revising a parent through its registered changes. S0 remains available alongside later records, allowing the next decision to relate a measured constraint to the original configuration. An unfavorable analytical score can still leave later-stage feasibility unresolved.

S1 denotes functional simulation, which tests whether the materialized candidate computes the required transform on the tested inputs. The execution path uses this result as a correctness prerequisite for subsequent evaluation. A functional failure cannot supply a valid performance outcome for that implementation. Its recorded status instead informs a subsequent supported choice or revision. The simulation gate and HLS resource gate answer different questions, so passing functional tests does not settle whether the candidate fits the task's resource limits. A study can execute simulation and HLS within one requested evaluation, so successive tool stages need not require separate model calls.

S2 denotes HLS synthesis and supplies period and resource estimates. Deterministic scoring applies the study's resource limits and computes its objective. Feasibility and performance then guide different decisions. An over-limit resource record identifies a constraint to address in a supported revision; the tool completed even though the candidate exceeds the task's limit. A feasible record provides an objective value for comparing evaluated configurations. In the hybrid family, changing concatenation alters parallelism through the generator's derived mapping. Testing the resulting child determines the actual resource and period tradeoff instead of assuming that analytical scaling carries through synthesis.

S3 physical implementation records routed timing, resources, build status, and bitstream availability. Its selective use shifts the decision from broad screening to whether a candidate remains viable after physical implementation. A routing or timing problem can motivate supported revision or deprioritization; a successful build can justify an enabled board evaluation. Operators schedule physical implementation in the current system.

S4 board measurement tests correctness and measures invocation rate under the requested protocol. For a board-rate objective, the agent uses this measured resource/rate tradeoff to reconsider earlier choices; HLS period or routed clock need not preserve the board ranking. Buffer choices depend on reduction, direction, and batch size, so the next decision must use evidence matching the task. Operators schedule board measurement; completed records can inform later choices, as in Section IV-E's board-conditioned follow-up. The enabled execution path constrains requests, and every candidate need not traverse all five stages.

A stage observation can come from an available record or from a newly executed permitted evaluation. The framework records the stage identity, completion status, parsed metrics when available, and failure or timeout information when execution does not complete. Failed or timed-out executions remain in the exploration history. An incomplete check establishes no correctness at that stage; earlier completed stage evidence remains attached to the candidate.

Returned observations update the state according to Eq. (2):

$$X_{t+1} = U(X_t, \hat{a}_t, o_t), \tag{2}$$

where U denotes the state-update operation, $\hat{a}_t$ is the accepted action, and o_t is the observation returned by the enabled execution path. The update adds materialized candidates to $\mathcal{C}_t$ while retaining eligible parents. The history H_t records requested actions and their outcomes, and E_t associates each visible observation with its candidate and stage, distinguishing a parameter change, a tool failure, and a measured resource violation. Budget state B_t accounts for charged requests, including rejected requests and evaluations without a feasible outcome. Equation (2) describes an accepted action and its returned observation; a rejection updates history and budget without adding a candidate or valid stage result. The next round distinguishes unqueried configurations from prior requests and can use a resource violation to select a supported modification. The state transition carries exploration history and visible evidence across successive logical rounds, including rounds executed through separate driver invocations.

Completed episodes use the fixed charged-request budget; summaries retain the best feasible observed outcome, or none when no evaluation is feasible. An explicit stop or defer action applies only to execution paths that expose it.

For example, in the 36-bit Barrett pilot of Section IV-D, the first proposal is a 64-butterfly-unit hybrid child (vertical size 8, horizontal size 4, concatenation factor 2) whose HLS record exceeds the LUT limit. The update records the observation and charges a request. In the next round, the feedback agent names that child as parent and lowers the concatenation factor from 2 to 1 while retaining vertical size 8, citing the LUT violation. Validation accepts the new signature; AutoNTT derives horizontal size 4, producing 32 butterfly units. The returned HLS result is within the limit.

IV. Evaluation

We evaluate four questions: whether later tool stages change the feasible design region, whether returned observations improve later decisions under a fixed request budget, whether the loop can execute fresh revisions and reuse its control flow with a second generator, and what additional tradeoffs appear after physical implementation and board measurement.

A. Evaluation Protocol and Experimental Setup

Replay covers 197 saved candidates in 12 settings ($N \in \{4096, 8192\}$, 32/48-bit moduli, Barrett/Montgomery/word-level Montgomery), with 36 episodes per agent arm. Fresh simulation/HLS uses Barrett/Montgomery: four 40-bit settings at $N \in \{2048, 4096\}$ and two 36-bit settings at $N = 2048$. The former has 16 distinct results including references/reuse; the latter has 11 configurations with simulation/HLS records across ten complete episodes. U280 studies cover 23 main implementations across eight settings and four controlled buffer implementations across two settings.

Replay charges logical requests, including invalid or duplicate outputs, without automatic retries; three consecutive transport errors stop execution as partial. Three requests expose at most 15.8–23.1% of each setting's 13–19 candidates. Feedback/no-feedback share the first model call and analytical candidates but retain separate action histories; only feedback sees its own tool observations. Analysis uses all saved measurements. Agent arms repeat three times per setting; numerical controls run once. Cap-first prioritizes analytical cap compliance before cycles. Family-calibrated rescales estimates using its own returned same-family median HLS period and HLS/analytical resource ratio. Initial values of 4 ns and 1 match Cap-first at budget one. The post-hoc cycle-only diagnostic ignores cap priority.

Fresh AutoNTT uses the same visibility rules and analogous controls, applying its HLS LUT/DSP gate to analytical estimates. The 36-bit pilot adds Latency-first, which ignores predicted caps but keeps measured feasibility. Each method receives two requests; invalid, duplicate, and reused-configuration requests consume budget.

Replay and fresh AutoNTT use the HLS-estimated compute-time proxy $J_{\mu s} = C_{\mathrm{ana}} \tau_{\mathrm{HLS}} / 1000$, where C_{ana} is analytical cycle count and τ_{HLS} is HLS period in nanoseconds. Board invocation rate requires separate measurement. Replay conditional regret is $(J_b - J^*_{\mathrm{cat}})/J^*_{\mathrm{cat}}$, where J_b is the best feasible proxy within b requests and J^*_{cat} the setting's feasible catalog optimum. The regret mean excludes episodes without a feasible observation; feasibility counts report them separately. Replay feasibility requires HLS completion within 50% of configured capacities: effective limits are 260k lookup tables (LUTs), 520k flip-flops (FFs), 1500 digital signal processing (DSP) blocks, 250 18-Kb block RAMs (BRAMs), and 64 UltraRAMs (URAMs). Resource ratios take maximum utilization across configured capacities; analytical ratios additionally include off-chip port usage normalized to 16 ports. These are protocol limits, not device capacities. Fresh AutoNTT feasibility requires simulation/HLS success within 260k LUTs and 1500 DSPs.

Agent studies use Gemma-4-31B, temperature 0.2, high reasoning effort, and a 32,768-token limit for reported replay, corrected fresh runs, and NGen. Replay/36-bit/NGen use an OpenAI-compatible DingLabs endpoint; 40-bit uses Amazon Bedrock. Calls have no fixed API seed. Compile/simulation/HLS limits are 900/900/3600 s for 40-bit and 180/180/1200 s for 36-bit. The pilot exceeded its original 7200-s wall limit; later tool runs completed frozen requests with zero new model calls or policy rounds.

Physical studies run on one Alveo U280 with Xilinx Runtime (XRT) 2.16.204 and target a 250-MHz kernel clock. We build designs with TAPA [17] and Vitis/Vivado 2023.2 for the `xilinx_u280_gen3x16_xdma_1_202211_1` platform. For batch size m and transform direction d, board rate is $R_d(m) = m/\operatorname{median}_i T_{i,d}(m)$ over 30 technical repeats, where $T_{i,d}(m)$ is kernel-invocation time. Primary comparisons use $m = 128$ transforms per invocation. This repeated-input invocation rate excludes host-process overhead and is not independent-request service throughput. U280 pairs report kernel-level synthesis/build resources; whole-device routed

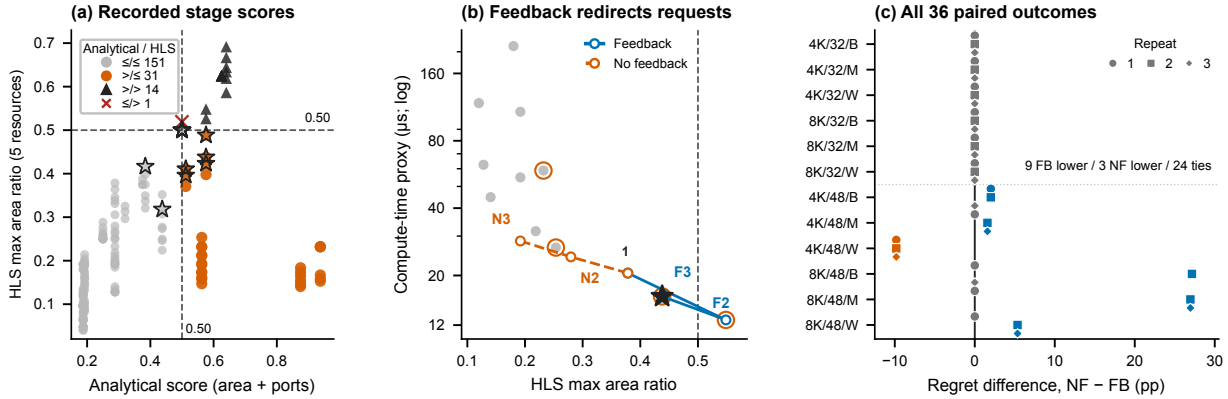


Fig. 2. Feedback-guided exploration. (a) All 197 candidates; dashed lines mark the 0.50 caps and stars the 12 catalog optima. Legend pairs give analytical/HLS cap compliance. (b) Largest feedback reduction in final regret: $N = 8192$, 48-bit Barrett, repetition 2; all 14 measured candidates. Rings: analytical-over-cap; line: HLS cap; star: optimum. Labels 1, F2/F3, and N2/N3 denote shared-first, feedback, and no-feedback requests. (c) All 36 paired regret differences; positive favors feedback. Settings: N (4K/8K), modulus bits, reduction (B/M/W: Barrett/Montgomery/word-level Montgomery). Markers denote repetitions 1–3 with vertical offsets.

counts have a different scope.

AutoNTT exports a candidate list (the exported search trace) and selects a configuration (the generator-selected reference). Board pairs compare it with supported alternatives inside/outside the trace. Buffer comparisons fix the configuration: split uses per-stream tasks; combined uses one task for partitioned multistream storage. Two sessions use the same card, XRT version, and shell. Table III uses within-session comparisons: session one for pairs 1/3, session two for 2/4–6 including the repeated 32-bit Montgomery pair. The Barrett follow-up remains separate.

B. Cross-Stage Evidence Changes the Useful Search Region

Analytical cap compliance alone excludes part of the HLS-feasible region. Figure 2(a) shows 31/197 candidates (15.7%) above the analytical cap of 0.50 but within the HLS cap. Of these crossings, 24 arise from port usage, which only the analytical ratio includes; seven reflect different analytical/HLS BRAM-count conventions.

An offline sensitivity check removes the analytical port term and replays Cap-first and Family-calibrated using HLS's five area resource types. Only two Family-calibrated third requests change; both rules retain the same best feasible proxy and regret at every budget in all 12 settings. Neither model arm was rerun with these inputs.

Fresh implementations also show LUT disagreement beyond the replay port convention. The four 40-bit generator-selected references meet the analytical LUT/DSP cap but exceed the HLS LUT gate, with HLS-to-analytical LUT ratios of 1.519–1.819×. Across all 14 matched designs, this ratio spans 1.136–1.819×, while DSP counts match. HLS LUTs exceed Vivado kernel-synthesis LUTs by 1.321–1.641× across ten matched designs. Four kernel-synthesis/analytical pairs have LUT ratios of 0.777–0.987×, reversing the discrepancy's direction. These reversals motivate retaining stage-specific resource observations instead of imposing one stage's feasibility ranking on later decisions.

TABLE I

FINITE-CATALOG REPLAY. FEAS.: FEASIBLE EPISODES; REGRET: CONDITIONAL MEAN REGRET (%); OPTIMA: CATALOG OPTIMA AFTER THREE REQUESTS. AGENT ARMS USE THREE REPETITIONS/SETTING; NUMERICAL CONTROLS RUN ONCE.

	Budget 1		Budget 2		Budget 3		
Decision method	Feas.	Regret	Feas.	Regret	Feas.	Regret	Optima
LLM, feedback	34/36	17.518	36/36	7.720	36/36	3.223	29/36
LLM, no feedback	34/36	17.518	36/36	10.840	36/36	5.154	23/36
Cap-first	11/12	11.873	12/12	10.884	12/12	5.254	7/12
Family-calibrated	11/12	11.873	12/12	6.956	12/12	5.703	6/12
Cycle-only (post hoc)	4/12	2.453	9/12	1.090	11/12	0.892	10/12

C. Feedback-Guided Decisions under a Fixed Budget

Feedback improves configuration selection within the same three-request budget: conditional mean regret falls from 5.154% to 3.223%, a 37.5% relative reduction, and catalog-optimum episodes increase from 23/36 to 29/36 (Table I). Feedback yields lower/higher/equal regret in 9/3/24 paired episodes. Gains concentrate in the 48-bit settings; all 32-bit episodes reach the optimum in both arms.

Across 108 selections per arm, feedback selects analytically-over-cap but HLS-feasible candidates 14 times, versus two without feedback. All nine improved episodes obtain their best result from such a candidate, absent from the matched no-feedback selections. This selection pattern links the regret gains to evaluating feasible designs that analytical cap priority would deprioritize.

The generator-selected 80-butterfly-unit hybrid reference (S0 ratio 0.625) has no HLS record and is absent. In Fig. 2(b), feedback follows the shared 32-unit hybrid with a faster-cycle 48-unit hybrid that fails the HLS cap. Request three reaches the optimum with another 32-unit hybrid sharing the failed probe's analytical ratio but satisfying HLS feasibility. No-feedback chooses a 24-unit hybrid then a 16-unit iterative configuration. Panel (c) shows all 36 outcomes; all three feedback losses occur in 4096-point 48-bit word-level Montgomery.

The post-hoc cycle-only diagnostic reaches 10/12 catalog optima at budget three, with 0.892% conditional regret on 11/12 feasible settings. Both agent arms find feasible outcomes

TABLE II

36-BIT BARRETT REVISION ($N = 2048$, TWO CHARGED REQUESTS/METHOD). THE FIRST 64-BU PROPOSAL IS SHARED; REMAINING ROWS SHOW REVISIONS. BU: BUTTERFLY UNITS; HLS GATE: 260K LUTS/1500 DSPS; $J_{\mu s}$: HLS-ESTIMATED COMPUTE-TIME PROXY (μS).

Method / request	BU	HLS LUT	DSP	$J_{\mu s}$	Gate
Shared by four methods	64	280,686	704	2.605	fail
LLM, feedback	32	140,129	352	3.909	pass
LLM, no feedback	80	329,936	880	2.873	fail
Cap-first (split)	64	314,832	704	3.042	fail
Family-calibrated	48	224,012	528	3.284	pass

TABLE III

U280 ALTERNATIVES (UPPER ROWS) VERSUS REFERENCES ($N = 4096$, $m = 128$). BU: BUTTERFLY UNITS; FWD/INV: FORWARD/INVERSE INVOCATION-RATE CHANGE. KERNEL-SYNTHESIS LUT/DSP; CLOCK FROM BINARY METADATA. †: NEGATIVE POST-ROUTE SLACK AT 250 MHZ. PAIRS 5/6 REUSE REFERENCES 2/4.

			Kernel		Kernel	Rate change (%)	
#	Workload	Family / BU	LUT (k)	DSP	clock (MHz)	FWD	INV
Trace alternatives versus generator-selected references							
1	32-bit Barrett	Dataflow / 96	167.8	1056	250.0	+10.05	+13.75
		Hybrid / 128	322.2	1408	233.3†		
2	32-bit Montgomery	Dataflow / 96	159.4	1056	250.0	+22.00	+33.04
		Hybrid / 128	297.7	1408	189.7†		
3	48-bit Barrett	Dataflow / 96	238.1	1632	208.5†	+60.29	+77.71
		Hybrid / 64	219.3	1088	233.4†		
4	48-bit Montgomery	Dataflow / 96	234.6	1632	170.5†	+48.69	+63.59
		Hybrid / 64	225.7	1088	213.7†		
Supported resource-saving hybrids outside the exported trace							
5	32-bit Montgomery	Hybrid / 64	205.5	704	217.2†	−3.27	−4.40
		Hybrid / 128	297.7	1408	189.7†		
6	48-bit Montgomery	Hybrid / 32	149.3	544	250.0	−4.23	−5.68
		Hybrid / 64	225.7	1088	213.7†		

in all 36 episodes at budget three; the diagnostic's lower conditional regret therefore has less coverage.

D. Fresh Configuration Exploration and Revision

Ten distinct 40-bit proposal results satisfy the HLS resource gate. We measured the four generator-selected references after policy rounds; no method received these measurements as feedback. Against these HLS-infeasible references (Section IV-B), each setting's best feasible proxy is 6.7–64.9% larger. Feedback/no-feedback follow the same two-step trajectory, and numerical controls reach the same best feasible values. The framework thus constructs HLS-feasible alternatives to references that exceed the resource gate.

The 36-bit pilot's parents exceed 260k analytical LUTs; proposals use at most 96 butterfly units. All Barrett methods except Latency-first (96 units) share a 64-unit hybrid child: its 180,960 analytical LUT estimate meets the cap, but HLS exceeds it. Using the returned violation, the agent revises the child from 64 to 32 butterfly units and recovers HLS feasibility (Table II), trading a larger compute-time proxy for resource compliance.

Family-calibrated achieves a lower Barrett proxy than feedback and alone finds a feasible Montgomery configuration. LLM without feedback, Cap-first, and Latency-first find no feasible result in either workload.

Second-generator validation. NGen [3] uses natural-order forward cyclic NTTs (modulus $q = 12289$, $N \in \{256, 512\}$),

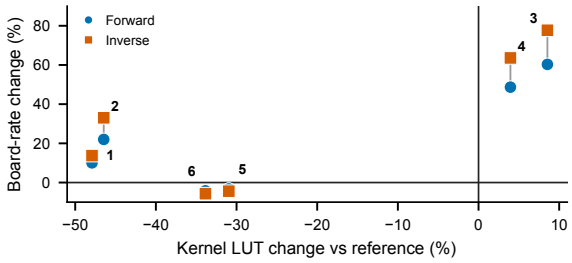


Fig. 3. The six U280 pairs in Table III: x is alternative-versus-reference kernel LUT change and y is board invocation-rate change; circles/squares denote forward/inverse transforms and pair numbers match the table.

eight configurations per task (2/4 lanes, 1/2 processing elements, Barrett/Shoup), three repeats/task and three requests/arm, sharing a 2-lane, 1-element Barrett reference. All arms see cycle hints matching simulated completion cycles; feedback and the numerical rule also see queried observations. The rule scales reference resources by lane/element counts, then ranks by hints. Feasibility requires correctness within a preflight-selected 5000-cell limit in Yosys 0.50 generic synthesis, distinct from FPGA/HLS metrics. We minimize first-input-to-final-output cycles.

The 54 charged slots yield 37 RTL evaluations over ten configurations. Feedback/no-feedback reach feasible catalog optima in 5/6 and 0/6 episodes at budget three; the numerical rule reaches 6/6 by budget two. Two HTTP 429 failures per model arm and one additional no-feedback interruption (unknown provider outcome) consume budget. These runs reuse the same agent–validation–tool-feedback loop for RTL configuration selection and evaluation with backend-specific parameters and tools.

E. Physical Evidence Exposes Additional Tradeoffs

Board measurements identify higher-rate configurations and resource-saving alternatives within the supported design space. Across six U280 pairs (Table III, Fig. 3), trace dataflow alternatives raise invocation rate, reducing kernel LUTs at 32 bits but using more at 48 bits. Smaller Montgomery hybrids outside the trace trade modest rate losses for fewer LUTs and half the DSP count.

In 48-bit Montgomery, dataflow has a lower saved kernel clock than the hybrid reference (170.5 versus 213.7 MHz) yet raises forward/inverse invocation rate by 48.69/63.59%. Retaining board-rate observations lets exploration prioritize this higher-rate option despite its less favorable intermediate clock.

At fixed $N = 4096$, 40-bit workload, architecture, and 32-butterfly-unit parallelism, combined buffering lowers HLS period from 7.347 to 6.316 ns and reduces kernel-synthesis LUTs for both reductions. The full 24 board ratios over six batch sizes and two directions (Fig. 4) show a different response: for $m \geq 128$, combined/split rate ratios fall to 0.939–0.958 for Barrett but remain within 0.6% of one for Montgomery. For large-batch Barrett, split buffering preserves invocation rate; for Montgomery, combined buffering saves LUTs with near-equal rate.

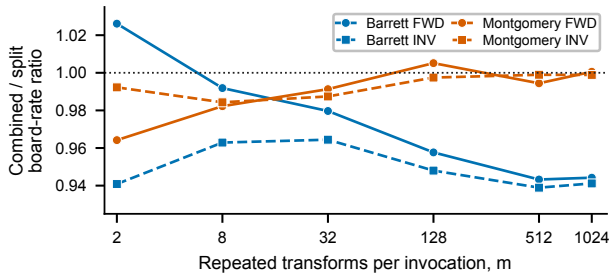


Fig. 4. Full controlled buffer sweep. Each curve is the combined/split board-rate ratio over $m \in \{2, 8, 32, 128, 512, 1024\}$; the dotted line denotes equal rate. The board response differs by reduction, direction, and batch size.

The separate 48-bit Barrett follow-up tests selection of one informative board measurement using completed observations. Operators measure the selected candidate. A predefined rule maximizes the butterfly-unit fold change from a measured same-family design and selects the higher-rate candidate; the model selects it in one of three calls.

V. Discussion and Limitations

Staged feedback supports two complementary forms of design improvement: selecting better configurations within a fixed request budget and revising configurations to recover feasibility. Replay demonstrates the selection benefit, while the fresh Barrett revision demonstrates feasibility recovery. NGen carries the same agent decision loop into RTL evaluation. U280 measurements provide the resource–invocation-rate tradeoffs needed to select implementations for different task objectives.

Replay optima remain finite-catalog results; model repetitions do not add independent workloads. Board evidence covers tested inputs on one U280 under Section IV-A's protocol, leaving all-input correctness and independent-request throughput unverified. We have not systematically evaluated policies conditioned on routed implementation results.

Future work will extend the agent's supported choices to compatible combinations of architecture, modular-arithmetic, buffering, and evaluation components for task-specific requirements. Such composition requires matched modulus, roots, ordering, normalization, and interfaces. The shared control/state flow validated with NGen provides a starting point for integrating these backend-specific components.

VI. Conclusion

We present an agentic design-exploration framework for generator-supported FPGA NTT accelerators. A stateful Design Exploration Agent uses stage-specific evidence to decide which supported configuration to evaluate, revise, or promote next. Deterministic validation grounds each request, generator components construct fresh implementations when needed, and the enabled tools return observations to the Exploration State. The next decision uses the updated state, including the candidate history and remaining request budget.

Finite-catalog replay shows that the agent lowers conditional mean regret for the compute-time proxy relative to the two predefined numerical rules and its own no-feedback ablation under a three-request budget. A fresh generator-backed revision recovers feasibility after an HLS resource-limit violation. Separately, operator-scheduled U280 measurements expose resource and invocation-rate tradeoffs that intermediate timing metrics do not capture. NGen validates the same agent–validation–tool-feedback loop with a second generator while retaining backend-specific parameters, tools, and metrics. Backend-specific adapters extend the same agentic exploration loop across HLS and RTL implementation flows.

References

[1] D. Kumarathunga, Q. Hu, and Z. Fang, "AutoNTT: Automatic architecture design and exploration for number theoretic transform acceleration on FPGAs," in *Proc. IEEE 33rd Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*. Fayetteville, AR, USA: IEEE, 2025, pp. 1–9.

[2] F. Krieger, F. Hirner, A. C. Mert, and S. Sinha Roy, "OpenNTT - an automated toolchain for compiling high-performance NTT accelerators in FHE," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*. New York, NY, USA: ACM, 2024, pp. 1–9.

[3] virtualsecureplatform, "NGen: Streaming number theoretic transform hardware generator," https://github.com/virtualsecureplatform/NGen, 2026, source revision dbb4117, accessed September 29, 2026.

[4] Y. Yang, S. R. Kuppannagari, R. Kannan, and V. K. Prasanna, "NTTGen: A framework for generating low latency NTT implementations on FPGA," in *Proc. 19th ACM Int. Conf. Comput. Frontiers (CF)*. New York, NY, USA: ACM, 2022, pp. 30–39.

[5] F. Hirner, A. C. Mert, and S. Sinha Roy, "Proteus: A pipelined NTT architecture generator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 7, pp. 1228–1238, 2024.

[6] B. C. Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present, and future," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2628–2639, 2020.

[7] A. Sohrabizadeh, C. H. Yu, M. Gao, and J. Cong, "AutoDSE: Enabling software programmers to design efficient FPGA accelerators," *ACM Trans. Design Autom. Electron. Syst.*, vol. 27, no. 4, pp. 1–27, 2022.

[8] L. Du, T. Liang, S. Sinha, Z. Xie, and W. Zhang, "FADO: Floorplan-aware directive optimization for high-level synthesis designs on multi-die FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*. ACM, 2023, pp. 15–25.

[9] C. Lo and P. Chow, "Multi-fidelity optimization for high-level synthesis directives," in *Proc. 28th Int. Conf. Field Programmable Logic and Applications (FPL)*, 2018, pp. 272–279.

[10] Q. Sun, T. Chen, S. Liu, J. Chen, H. Yu, and B. Yu, "Correlated multi-objective multi-fidelity optimization for HLS directives design," *ACM Trans. Design Autom. Electron. Syst.*, vol. 27, no. 4, pp. 1–27, 2022.

[11] H. Wang, X. Wu, Z. Ding, S. Zheng, C. Wang, N. Prakriya, T. Nowatzki, Y. Sun, and J. Cong, "LLM-DSE: Searching accelerator parameters with LLM agents," 2025, arXiv:2505.12188v3.

[12] R. Li, J. Xiong, and X. Wang, "iDSE: Navigating design space exploration in high-level synthesis using LLMs," 2025, arXiv:2505.22086v2.

[13] V. Sharma, X. Fu, J. Haris, and J. Cano, "Towards autonomous accelerator design: FPGA accelerator generation with SECDA," 2026, MLArchSys 2026; arXiv:2606.11117v1.

[14] N. Mashnoor, M. Akyash, H. Kamali, and K. Azar, "TimelyHLS: LLM-based timing-aware and architecture-specific FPGA HLS optimization," in *Proc. IEEE Int. Conf. Omni-layer Intell. Syst. (COINS)*. IEEE, 2025, pp. 1–6.

[15] A. E. Oztas and M. Jelodari, "Agentic-HLS: An agentic reasoning based high-level synthesis system using large language models," 2024, AI for EDA Workshop, NeurIPS 2024; arXiv:2412.01604v2.

[16] H. Wu, Z. He, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "ChatEDA: A large language model powered autonomous agent for EDA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 10, pp. 3184–3197, 2024.

[17] L. Guo, Y. Chi, J. Lau, L. Song, X. Tian, M. Khatti, W. Qiao, J. Wang, E. Ustun, Z. Fang, Z. Zhang, and J. Cong, "TAPA: A scalable task-parallel dataflow programming framework for modern FPGAs with co-optimization of HLS and physical design," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 4, pp. 1–31, 2023.