International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Automatic Model Selection Based on Task Complexity

Mr. Lakshminarasimhan Santhanam

CEO, Founder, QML, QAF Lab India

Abstract

Which solver should handle a given task — with which tools, how much verification, and what opportunity to escalate? Every AI application answers that question, and most answer it implicitly. Defaulting to a large language model spends reasoning on work that never needed it. Choosing a small model on price alone does not remove the cost; it relocates the cost into errors and rework. This paper sets out an Adaptive Intelligence Router that chooses among deterministic programs, retrieval and service interfaces, small language models, more capable language models, and accountable human review. A task's demands are carried by a complexity profile in which every component has a defined role: three dimensions generate the admissible set, one fixes the reliability floor and the mandatory oversight level, and the rest are predictive features and nothing more. Selection then becomes a constrained sequential decision problem spanning routing, verification, retries and handoffs. I distinguish the pre-execution success probability that chooses an action from the post-verification probability that licenses accepting its output, and show that only the second supports a bound on residual error among accepted work. An analytical illustration then parameterises a two-stage cascade by verifier recall and false-flag rate instead of escalation fraction. Doing so exposes a cost-quality frontier whose cheapest point is also its least accurate. The evaluation protocol proposed here compares static routing, fixed cascades, learned routing, decomposition, and a non-routing caching baseline under quality, latency and residual-risk constraints, and it counts the cost of certifying the router itself. The paper reports a design and a protocol; no router was trained and no benchmark results are given.

Keywords: model routing; adaptive inference; task complexity; model cascades; selective prediction; conformal risk control; learning to defer; verification; AI infrastructure; human oversight.

1. Introduction

An application that can choose between execution strategies needs to know which of them will finish the task in hand to the required standard, at the lowest total cost it is willing to spend on it. That is a different question from which strategy is the most capable, and the two come apart in ordinary cases. A two-line request can hide a genuinely hard piece of reasoning; a forty-page one can be almost entirely repetitive extraction. Input tokens, model parameters and participating agents are all easy to count, and none of them separates the first case from the second.

The allocation policy I adopt is cost minimisation subject to a reliability floor: ordinary computation where it settles a well-specified problem, a validated smaller model where its measured performance clears the floor for the task, and more capability or more oversight where it does not. As a principle this is uncontroversial, and on its own it decides nothing, because everything operational sits in the floor. What counts as adequate for a given task, who is entitled to set that level, what evidence shows that a

configuration meets it, and what should happen when no available configuration does — the principle settles none of it. An acceptance contract settles it, and until one is written the policy has no content. Specifying that contract, the routing policy that runs against it, and the experiments that would test both is the work of the rest of this paper.

2. Related work and contribution

None of this is new ground. FrugalGPT builds learned cascades that adapt the sequence of language models to the query at hand [1]; RouteLLM learns to route between a stronger and a weaker model from preference data [2]. Both motivate task-dependent allocation. Neither is evidence for the architecture proposed here, because what they report is tied to particular models, particular datasets and particular evaluation methods, and none of those conditions holds here.

Dekoninck, Baader and Vechev bring routing and cascading into a single frame and derive optimal strategies under their assumptions, with quality estimators doing much of the work [3]. A one-shot router has to decide before it sees an answer; a cascade can look at what came back. I take the sequential view from them without claiming that the operational policy below inherits their optimality. It does not. An approximation running on uncertain predictors, tools that change underneath it, and human handoffs is a different object, and it needs its own evaluation.

Selective prediction is the second foundation. Chow set out the error-rejection tradeoff that sits underneath every abstention rule [15]; SelectiveNet carries the joint treatment of prediction and rejection into deep models and makes the coverage-error trade-off explicit [4]. In an enterprise router, abstaining is not silence. It becomes clarification, retrieval, escalation, or a referral to a person. This means the system has to report how much work it resolves on its own next to the error rate of those resolutions. Accuracy bought by referring almost everything is not automation; it is a referral desk reporting good statistics.

The reliability of the routing signals themselves belongs to the calibration literature. Guo et al. document miscalibration in neural networks and the methods that improve confidence estimates [5]. Kadavath et al. find real self-evaluation ability in language models, and also the point at which it stops generalising to new tasks [6]. Zheng et al. catalogue the biases of language-model judges [7]. Read together, these are a warning against building on a solver's stated confidence, and an argument for acceptance outcomes measured independently of it.

The third foundation is distribution-free risk control, and it bears on the guarantee this architecture needs more directly than the other two. Conformal methods turn a heuristic score into a decision rule with finite-sample validity, and they do it without requiring the score to be correctly specified. Learn-then-Test casts risk control as a multiple-testing problem [10]; conformal risk control extends coverage guarantees to the expected value of monotone loss functions [11]. What this router must satisfy is residual error below a stated level among the outputs it accepts. That is an expected-loss constraint on a selected subpopulation, which is exactly the shape those results address. Recent work has already pointed the machinery at this setting. Conformal Cascade uses conformal set size as the deferral rule across inference tiers and obtains a per-tier accuracy guarantee [12]. PASC handles joint coverage across the stages of a multi-stage pipeline [13]. RouteNLP couples conformal cascading with distillation in a deployed router [14]. Guarantees of this form are therefore available. Section 4.3 adopts them as the calibration layer for the acceptance rule, and treats their assumptions as conditions a deployment must monitor rather than properties it may take for granted.

Referral to a person has an established formal treatment too, and this design should inherit it rather than reinvent it. Learning to defer generalises the reject option toward a particular expert whose accuracy varies with the input: Mozannar and Sontag give a consistent surrogate loss by reduction to cost-sensitive learning [16], and Verma and Nalisnick correct the calibration of the resulting estimates of expert correctness [17]. In Section 5.1 a referral action is therefore scored against the reviewer's measured accuracy on the relevant input type, on the same footing as any automated configuration in the portfolio.

Learning from execution logs runs into a missing-data problem, since ordinarily only the chosen solver leaves an observable outcome behind. Dudik, Langford and Li develop doubly robust methods for policy evaluation and learning under contextual-bandit feedback [8], and such methods inform how a routing policy can be assessed from historical decisions, subject to overlap, adequate logging, and the estimator's own assumptions. What they cannot do is recover evidence about solvers that were never sampled in the relevant contexts. Section 4 also poses the objective as a constrained Markov decision problem, where the classical treatment says when such problems admit optimal stationary policies and when randomisation between policies becomes necessary [18].

Moslem and Kelleher's 2026 survey organises the routing and cascading literature across difficulty, preferences, uncertainty, multimodality and several further paradigms [9]. My claim is correspondingly narrow: taking heterogeneous solver selection, calibrated acceptance, workflow decomposition and accountable stopping rules, designing them jointly, and evaluating the result as one system, including what it costs to keep that system running.

2.1 Scope of the proposed contribution

I claim five contributions. A complexity representation in which every dimension has an assigned role in selection. A constrained policy that selects execution configurations instead of model names. An explicit separation between the probability used to choose an action and the probability required to accept its output, together with the population guarantee that follows from the second. A decomposition procedure with a structural validator and a rule for allocating error budgets. And an evaluation protocol that counts the cost of certifying the router itself. These are design contributions. The paper is conceptual and methodological; the experimental work it specifies remains to be done.

3. Representing task demands

Write x for the request together with its accessible context, its input artifacts, and the outcome being asked for. Its initial complexity profile is C(x), set out in Equation (1). I selected these eight dimensions because each one can change the selection on its own; candidates that only moved with another dimension were dropped. Nothing here establishes that this is the right set, and Section 12.2 says so. Complexity in this paper means operational demand on a solver, in the narrow sense the dimensions below define. The term carries no claim about intelligence, human or machine.

$$C(x) = (R, A, K, L, T, V, S, U) \qquad (1)$$

Table 1. Complexity dimensions, their role in selection, and observable evidence

Dimension	Role	Interpretation	Candidate observations
R Reasoning	Predictive	Dependency and planning demand	Number of dependent steps; branching; constraints to satisfy

Dimension	Role	Interpretation	Candidate observations
A Ambiguity	Predictive	Unresolved interpretation	Missing units, definitions, objectives, or conflicting instructions
K Knowledge	Constraint	Specialist evidence required	Domain specificity; need for current or private sources
L Context	Constraint	Input handling burden	Tokens, documents, cross-document links, retrieval coverage
T Tools	Constraint	Execution and action demand	Calls, tool dependencies, permissions, reversibility
V Verification	Predictive	Difficulty of establishing success	Availability of exact tests, reference answers, expert rubrics
S Stakes	Floor	Consequence of an error	Exposure, affected parties, reversibility, required accountability
U Uncertainty	Predictive	Initial lack of reliable evidence	Out-of-distribution features; poor input quality; weak calibration

3.1 The role of each dimension

A profile that is only recorded changes nothing. In Version 1.0 the vector was defined carefully and then appeared in no later equation, so nothing in the selection procedure depended on it. Each dimension now carries exactly one of three roles, and Sections 4 through 8 use it in that role.

K, L and T are the constraint-bearing dimensions. Together with the requirement record they settle feasibility and generate the admissible set directly, as Equation (2) shows: a configuration missing a required modality, a tool permission, context capacity, or data-handling authority drops out before any cost is compared. Each is evaluated as a predicate, and none of them contributes a weight to the objective in Equation (7).

S is floor-setting. It fixes the reliability threshold and the mandatory oversight level, per Table 2. It stays out of the cost objective, and it cannot be traded against price.

R, A, V and U are predictive features for the estimator in Section 8. They shape which admissible configuration gets chosen, and they enter only after Equation (2) has fixed what is admissible at all.

This partition exists to prevent one common and specific mistake: folding stakes into a scalar index alongside reasoning depth and averaging the two. High reasoning demand calls for more capability; high stakes call for more oversight. Those are different responses, and a single number cannot hold both. The partition also disciplines what a score is allowed to mean. A reasoning-depth value of 0.8 is not a measurement unless the scoring procedure, the annotations and the inter-rater agreement behind it have been defined. A constraint dimension needs no such scale at all, since it is evaluated as a predicate against the requirement record.

Raw measurements should survive alongside any normalised representation. Scale continuous features on the training split; give ordinal labels written anchors. Missing values need explicit indicators (an

unknown risk level must never decay quietly into a low one), and under the partition above a missing constraint dimension renders a configuration inadmissible rather than admissible by default.

The requirement record supplies the operands the constraint dimensions test against: modality, output schema, numerical precision, source freshness, privacy restrictions, allowed tools, monetary budget, deadline. A text-only model does not become eligible for direct image interpretation because its other scores look good. A capable external model is excluded outright when the data-handling policy allows only local processing.

$$A(x,h) = \{\, a : \mathrm{feas}(a \mid K, L, T, \rho) \;\wedge\; O(a) \geqslant O_{\min}(S) \,\} \qquad (2)$$

In Equation (2), ρ is the requirement record, feas the conjunction of feasibility predicates over the constraint dimensions, O(a) the oversight a configuration actually provides, and $O_{min}(S)$ the oversight its stakes tier demands. Admissibility is settled before cost, and this equation is the only door through which a configuration enters consideration.

Table 2. Stakes tiers, reliability floors, and mandatory oversight

Tier	Character	Reliability floor τ	Mandatory oversight	Residual-risk budget ε
S0	Internal, reversible, low exposure	0.90	None	0.10
S1	Customer-visible, reversible	0.95	Sampled post-hoc audit	0.05
S2	Customer-visible, costly to reverse	0.99	Named reviewer before release	0.01
S3	Irreversible, regulated, or privileged	Not applicable	Accountable approver; automation may draft only	No automated acceptance

The values in Table 2 are illustrative defaults for exposition and carry no standing as a safety standard. Two features of the table are structural rather than illustrative. The application owner fixes the levels before cost optimisation begins, and S3 is an absolute bar rather than a very high threshold. I set it that way deliberately: at that tier no reliability estimate authorises automated acceptance, whatever its value, and the router prepares work for a person who remains accountable for it.

Rather than invent scores, the examples below follow the partition. Invoice extraction becomes a candidate small-model task only once layout, OCR quality, and the cost of a wrong identifier have been weighed. Comparing termination clauses across a set of contracts pushes up L and K, because it needs evidence coverage and precise clause alignment. A migration plan for banking systems is dominated by S and T and hardly at all by R. Section 11.1 returns to that case and works through the routing decision it produces.

Keep the first profiler cheap: metadata, schema validation, token counts, rules, with a learned classifier held back for the cases those leave unresolved. Calling an expensive model to decide whether a task needs an expensive model defeats the economics the router exists to deliver. Profiler cost and profiler error therefore stay visible in the evaluation, carried in the routing overhead of Section 9.

4. Formal selection objective

4.1 Actions, history, and outcome

I define a solver action a as a whole execution configuration rather than as a model: an implementation or model version, its prompting or program configuration, its tools, its context strategy, and a proposed verifier. Naming a model names only one component of the thing whose performance is actually measured. One configuration might invoke a deterministic program, another a language model, another a human workflow. Parameter count does not order these by capability. A specialised smaller model can beat a general one on a narrow workload, and a human review queue has service-time and capacity behaviour that no inference endpoint shares.

Let h be the observable execution history — completed actions, outputs, tool events, verification evidence. Let A(x,h) be the admissible set from Equation (2). Let Y be a binary reference outcome recording whether the final deliverable satisfies its task contract. The router estimates the probability that Y = 1 from data; Y itself is observed only after the fact, and sometimes long after.

$$p_a(x,h) = P(Y=1 \mid x,h,a) \qquad (3)$$

Equation (3) holds for terminal configurations only. Take a non-terminal action instead — a clarification, a retrieval, a repair — and the probability of eventually satisfying the contract depends on what the policy does next, so no quantity of the form P(Y = 1 | x, h, a) is a property of a by itself. Non-terminal actions are scored by expected continuation value under the current policy, defined in Section 4.4, and the screening rule below does not reach them. Section 6.2 keeps the same distinction in operational form.

4.2 Screening among terminal configurations

For terminal configurations, a useful screening rule picks the cheapest admissible action whose validated reliability clears a screening level, using a conservative lower-bound estimate marked by the superscript LB. The rule decides what is worth attempting; whether the result may be accepted is a separate question, settled in Section 4.3.

$$a^* = \arg\min_a \{\hat{c}_a + \lambda_t \hat{l}_a\} \text{ subject to } p_a^{LB} \geq \tau_{\text{scr}}(x),\ a \in A(x,h) \qquad (4)$$

Here $\hat{c}_a$ is the predicted total completion cost for that configuration, $\hat{l}_a$ its predicted completion latency, and λ_t the factor converting latency into the same utility units as cost. The screening level in Equation (4) is an efficiency parameter: it stops the system spending a call on an attempt that probably will not work. The safety threshold is τ(S), which enters only at acceptance.

4.3 From a per-task threshold to a population guarantee

The screening rule in Equation (4) and the residual-risk constraint in Equation (8) are different objects, and the first does not imply the second. Equation (4) uses p_a, evaluated before anything executes. Acceptance happens afterwards, once a verifier has looked at the output. To condition on acceptance is to condition on the verifier having passed, and the passed set is not a random sample of the tasks routed to a: it over-selects outputs the verifier found convincing, including the ones it found convincing for the wrong reason. A floor on p_a therefore establishes nothing direct about error among accepted work. Treating $\tau = 1 - \varepsilon$ as though it closed that gap is the most consequential error available in this design, and Version 1.0 made it.

What does support the constraint is the post-verification success probability of Equation (5), where Z_a denotes the verifier's decision on the output that configuration a actually produced.

$$q_a(x,h) = P(Y=1 \mid x,h,a,Z_a = \text{pass}) \qquad (5)$$

Suppose the acceptance rule admits an output only when a valid lower bound on q_a meets the floor $\tau(S)$. Write the joint probability out by the tower property and bound the inner conditional pointwise on the accepted event: $P(Y = 0, \text{accepted}) = E[1\{\text{accepted}\}(1 - q_a(x,h))] \leq (1 - \tau)\, P(\text{accepted})$. Divide through by $P(\text{accepted}) > 0$ and Equation (6) follows. Set $\tau = 1 - \varepsilon$ and the residual-risk constraint is recovered exactly.

$$P(Y=0 \mid \text{accepted}) \leq 1 - \tau \qquad (6)$$

Two conditions carry the argument, and both are empirical rather than notational. The first: the lower bound on q must hold simultaneously across the accepted population, not pointwise for each request taken in isolation. A family of individually valid ninety-five percent intervals, selected by the very rule that decides acceptance, does not keep its level through that selection. Distribution-free risk control is the right machinery here, and what it supplies is a calibration procedure rather than an assumption [10, 11, 12]. The second: q must be estimated on a calibration population exchangeable with the deployed one, which is precisely what the domain shift of Section 8.2 disturbs. Both conditions are monitorable, and both can fail; Section 11.3 makes the monitoring, and the fallback to mandatory review, a condition of operating at all.

4.4 Policy objective and constraints

For cascades the decision variable is a policy π mapping histories to subsequent actions: clarification, verification, acceptance, referral. Equation (7) states the objective over the whole trajectory rather than over a single call.

$$J(\pi \mid x) = \mathbb{E}_\pi[\, C_{\text{total}} + \lambda_t L_{\text{total}} + \lambda_h H \mid x\,] \qquad (7)$$

H is a separately defined severity-weighted harm measure. It is not a second copy of the quality score, and its coefficient λ_h has to specify the unit conversion explicitly. Non-negotiable requirements around data, authorisation or human review stay as hard constraints in Equation (2); they do not become penalties that a lower cost can buy off. Where harm resists credible quantification, the requirement belongs in Equation (2) as an admissibility constraint; assigning it a monetary figure that cannot be defended only moves the problem into the objective.

$$P_\pi(Y=0 \mid \text{accepted}) \leq \varepsilon, \quad P_\pi(L_{\text{total}} > d) \leq \delta, \quad P_\pi(\text{accepted}) \geq \gamma \qquad (8)$$

I add the third constraint because without it a degenerate policy wins: refer every request to a human. Nothing is accepted automatically, so the conditional error rate is vacuous and the first two constraints hold trivially. As long as human cost stays bounded, that policy looks optimal. The coverage floor γ, together with the referral cost carried in the total-cost term of Equation (7), is what makes automation a requirement instead of an option, and it is why Section 10.3 reports coverage next to error.

An aggregate constraint can hide poor performance in one language, one document family, one risk class, and the mechanism is specific rather than incidental. A router selects on predicted success. Groups with thinner representation in the training logs get less reliable predictions. Then, depending on which way the miscalibration runs, those groups are either under-served — routed cheaply and failing more often — or cost-loaded, escalated when they did not need to be and completed more slowly. Both are disparities the routing itself produced, and an aggregate ε can see neither. Hence the constraints are stated per prespecified group.

$$P_\pi(Y=0 \mid \text{accepted}, g) \leq \varepsilon_g \quad \text{and} \quad P_\pi(\text{accepted} \mid g) \geq \gamma_g \quad \forall g \in G \qquad (9)$$

Fix the groups G before evaluation begins. Reporting by group is not enough; the constraint has to bind. A policy optimised against an aggregate will finance its average performance out of whichever group is cheapest to fail.

What results is a constrained Markov decision problem, and feasibility is not guaranteed. A workload may contain tasks for which no admissible configuration reaches $\tau(S)$ at any cost. When that happens the coverage floor and the residual-risk constraint are in conflict, and one of them has to be renegotiated with the application owner rather than quietly relaxed by the optimiser. Classical results characterise when problems of this form admit optimal stationary policies and when randomisation between policies is required [18]. I use that framing to state conditions, not to claim a solution method; the approximation actually proposed is in Section 6.2.

5. System architecture and execution policy

The Adaptive Intelligence Router keeps requirement definition, action selection, execution and verification apart. A capability registry holds tested task performance, modalities, context limits, tool compatibility, cost measurements, latency distributions and version identifiers, and is updated whenever an implementation changes. A provider label or a model tier is no substitute for measured capability. Section 8.4 deals with what it costs to keep that registry current, which turns out to be a first-order determinant of whether any of this is economical.

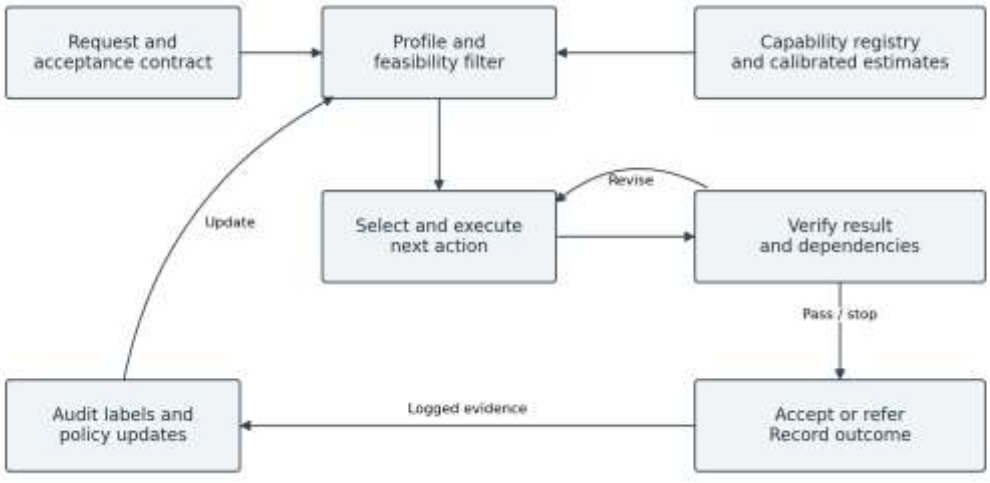

Figure 1. Proposed control loop. Revision can change tools, context, solver, or decomposition. Human referral is available whenever the contract or evidence requires it.

5.1 The solver portfolio

The portfolio spans deterministic programs for well-defined arithmetic and rules; database queries and retrieval interfaces for factual access; small models for validated extraction and classification; general models for synthesis and coding; more capable models for demanding reasoning or planning; and human specialists for judgment and accountability. Search and API results are inputs to be checked like any other input: freshness, coverage, permissions and returned evidence all have to be verified against the task contract.

I treat human review as a solver with an error model attached, and not as a backstop that is correct by definition. Its admissibility turns on availability as much as on competence: a referral is feasible only when expected queue wait plus service time fits inside the deadline and the reviewer pool sits below its utilisation cap. Referral capacity therefore belongs inside A(x,h) as a constraint, not merely in the objective as a cost. Routing work past that capacity does not complete it; it lengthens the queue, and the deadline is missed anyway. Where reviewer accuracy varies by input type, the deferral is scored against measured performance on that input type, in the manner the learning-to-defer literature makes precise [16, 17].

5.2 Conditional escalation

There is a familiar ladder: deterministic computation, then small models, then mid-tier, then frontier, then human review. As an account of typical cost order it is useful. As an ordering of capability it is wrong, and Section 4.1 has already said why parameter count will not do that job. The implementation is a graph over the admissible set, and the registry may well place a specialised small configuration above a general large one for some task family. The reasons for escalating vary too. A missing document calls for retrieval; ambiguous units call for clarification; a failed service call calls for recovery. None of these is repaired by a larger model, which can neither supply missing authorisation nor invent facts it does not have.

Starting cheap means starting with the lowest expected total-cost admissible strategy. It does not mean starting with the smallest model. The router can skip levels when a preliminary attempt looks unlikely to resolve the request, or when taking it would breach the deadline, and human review can be mandatory before execution even where the automated reasoning is trivial. Repeated attempts need three things: a stopping rule, a budget, and a preserved record of what failed.

6. Verification and acceptance

The task contract sets out required fields, acceptable evidence, tolerances, prohibited actions, and what counts as a successful final outcome. Deterministic arithmetic can use exact comparison or tolerance bounds. Extraction can use a schema, a source location and field-level reference labels. Code can use executable tests and invariants. Open-ended strategic analysis needs a rubric spanning evidence, alternatives, assumptions and internal consistency. Any single scalar quality value is inadequate if it lets polished prose offset a critical factual error.

Execution throws off observable signals: schema violations, failed tests, contradictory outputs, missing citations, unresolved dependencies, tool errors, assumption flags. None of these requires access to a model's private reasoning. A stated confidence of 0.98 may be logged as a candidate feature, but it is not a calibrated success probability, and the reliability predictor has to be tested against reference outcomes that owe nothing to that assertion.

Verification determines the behaviour of the whole system. Section 9 shows that both the achievable cost and the achievable error are set by the verifier's operating point, and that the cheapest configuration of the whole system is the one with the weakest verifier. The design question is therefore what recall can be attained on a given workload, and at what verification cost.

6.1 Avoiding circular verification

A verifier's pass decision is evidence about correctness. It is not correctness. Generation and judgment run through the same model and prompt family can reproduce the same error twice, and the second occurrence is then treated as confirmation of the first. Prefer exact checks wherever they exist, and

calibrate model judgments against independently assessed cases. For comparative evaluations, randomising answer order and using blinded reviewers removes avoidable judging effects; the research on language-model judges is clear that agreement with people does not make them infallible [7].

Independence between verifier and generator is what gives the calibration in Section 4.3 its meaning. Let generation and judgment share a model family, a prompt lineage and training data, and their errors correlate, q_a is overestimated, and the bound on accepted error fails — silently, and in the direction that hurts. Of the failure modes in this design I regard correlated verifier failure as the most dangerous, and I would look for it before anything else in a deployment review: it is the mechanism by which a system of this kind produces output that is confident, audited and wrong.

Let Z denote the verifier's decision. The operational quantity that matters is P(Y = 0 | Z = pass, accepted), estimated by independently auditing accepted outputs. Record false rejection as well, since it drives avoidable escalation and shows up directly as φ in Section 9.2. Outcomes that are unknown or delayed stay unlabelled until evidence arrives. Treating silence, or an absence of complaints, as success inflates the router's confidence in a systematic direction.

6.2 Proposed bounded execution procedure

1. Parse the request and fix the acceptance contract, the budget, the deadline and the required oversight. Derive the stakes tier from caller role and data class, as Section 11.2 describes, and not from the request text alone. Where a missing requirement would materially change the solution, ask.
2. Extract the inexpensive features, form A(x,h) by Equation (2), and identify any certified deterministic path. If a decomposition is proposed, put it through the structural checker of Section 7.1 before anything else.
3. For terminal configurations, predict cost, latency and pre-execution success, then apply the screening rule of Equation (4). For non-terminal actions, compare expected continuation value under the current policy instead. Choose the admissible action with the best estimate.
4. Execute inside the configuration's budget and permissions. Store the outputs, the source references, the tool events, and the exact version of the configuration that ran.
5. Verify the output, and verify every prerequisite it rests on. Write the test evidence and the calibrated reliability estimates back into the history.
6. Accept on three conditions and no fewer: the contract is satisfied, a valid lower bound on post-verification success q meets τ(S(x)) under the calibration of Section 4.3, and any mandatory oversight has occurred. Short of that, choose clarification, retrieval, repair, another solver, or referral.
7. Stop when no feasible continuation remains, when the budget is spent, or when the deadline forces a referral. Log the unresolved outcome and queue it for independent audit labelling.

The result is a policy skeleton that can be implemented directly. The optimal continuation value is generally unknown, so a first implementation might use measured stage success rates with a bounded search over a small portfolio. Whatever learned improvement follows has to beat that transparent baseline after paying for its own overhead.

7. Decomposing and routing workflows

Routing whole requests can hide chances to allocate capability locally. Take a request to analyse 50,000 customer complaints and recommend a strategic response. A great deal of that work is normalisation, extraction and aggregation; a much smaller part is interpreting trade-offs and choosing what to do. The

architecture represents work of this shape as a directed acyclic graph whose nodes carry typed inputs, output schemas, provenance, and local acceptance criteria.

7.1 Constructing and validating the graph

Decomposition has to come from somewhere, and where it comes from affects cost and risk both. There are two mechanisms. For task families the system recognises, the graph is a registered template, versioned in the capability registry alongside the configurations it references; its structure is fixed and only node-level routing varies. For requests it does not recognise, a model may propose a decomposition, in which case the proposal is itself a routed action, its cost lands in the routing overhead o of Section 9, and its output is not trusted.

A proposed graph is admitted only after a structural validator passes it. Node inputs and outputs must be typed. Every node's inputs must come from an upstream node or from x. The graph must be acyclic. Every field the acceptance contract requires must be produced by at least one node. Every node must also carry a local acceptance criterion drawn from the registry rather than written by the proposer. A proposer permitted to define its own success criteria has been handed the verification problem it was supposed to decompose. Proposals that fail validation are rejected without execution.

I restrict proposal to depth one: a node may not itself be decomposed. That bounds planning cost and keeps the action space finite. The price is whatever deeper structure the problem might have had. Lifting the restriction would need a termination argument and a bound on cumulative planning overhead, and I have neither.

Table 3. Illustrative allocation for complaint analysis

Work component	Candidate solver	Evidence required
Normalise and deduplicate	Rules and database operations	Record counts; duplicate policy; reversible lineage
Assign complaint categories	Validated small classifier or SLM	Stratified labelled sample; rare-category recall
Aggregate patterns	SQL or numerical programs	Reconciled totals; denominators; reproducible queries
Extract themes and examine causes	Small or general model with retrieval	Representative source excerpts; unsupported claims flagged
Synthesise strategic options	Lowest-cost configuration meeting the node contract	Traceable evidence; stated alternatives; cost and uncertainty
Approve business response	Accountable decision owner	Decision rationale; responsible owner; implementation limits

Table 3 is a candidate allocation, not a claim that each named tier will succeed. The synthesis row names a selection criterion rather than a tier. Naming the largest model there would reintroduce the assumption this architecture exists to test. A language model may well suggest causal hypotheses; complaint frequencies alone will not establish causes. Rare severe incidents also need explicit preservation,

because compressing 50,000 records into the largest clusters discards exactly the cases a decision maker needs to see.

7.2 Allocating local error budgets

Requiring local budgets to sum to the workflow budget leaves the division open, and the division has real cost consequences: a node where accuracy comes cheap should carry a smaller share than one where it comes dear. Let $c_j(\varepsilon_j)$ be the minimum cost of reaching failure probability ε_j at node j, decreasing and convex over the configurations available for that node. Minimise total cost subject to the workflow budget and Equation (10) gives the first-order condition.

$$-\frac{dc_j}{d\varepsilon_j} = \mu \quad \text{for all interior } j, \qquad \sum_j \varepsilon_j = \varepsilon_{\text{wf}} \qquad (10)$$

The marginal cost of error reduction equalises across nodes. Dividing the workflow budget uniformly across the m nodes is the baseline that condition should be required to beat. The corner cases matter more than the interior for deployment, since they are where the workflow turns out to be infeasible. If some node cannot reach any budget consistent with the total at any available cost, the workflow as specified is infeasible, and that node is where mandatory review or a redesign belongs.

7.3 Controlling propagated error

Where final success depends on all critical subtasks, strong average node accuracy can still leave workflow reliability weak. Ten required steps at 0.95 each give $0.95^{10} \approx 0.599$ under independence. Independence is rarely justified, since nodes share inputs, models and upstream errors, but the right response is to bound the quantity rather than discard the calculation. Hold the marginals fixed and the Fréchet–Hoeffding inequalities put the probability that all ten steps succeed somewhere in [0.500, 0.950]. At the upper limit failures align perfectly and always co-occur; at the lower they spread as widely as they can. The independent value of 0.599 sits inside that interval, and it is not a worst case.

$$P(\text{any critical failure}) \le \sum_j \varepsilon_j, \quad \text{provided } P(\text{failure}_j) \le \varepsilon_j \qquad (11)$$

That lower limit is the union bound of Equation (11). It needs no independence assumption, and it motivates the budget allocation of Section 7.2. Its weakness shows up in this very example: at ten nodes with $\varepsilon_j = 0.05$ it certifies only that the workflow succeeds at least half the time, which is not adequate assurance for consequential work. Tightening it takes fewer critical nodes, smaller node budgets, or a method that models the dependence directly — joint-coverage approaches for multi-stage pipelines address the last of these [13]. In either case the node probabilities have to be measured on the deployed upstream-input distribution, since what propagates is accuracy on the inputs a node actually receives rather than accuracy on clean ones.

The orchestrator keeps links from conclusions back to source records and reruns downstream nodes when an upstream artifact changes. A final verifier checks joins, denominator consistency, omitted segments and cross-node contradictions. Choose decomposition only when its expected savings or quality improvement exceed planning, coordination, verification and recomputation costs. Strongly coupled problems are often better handled as one reasoning unit. The treatment in Section 7.3 is incomplete in one respect: it bounds propagated error without modelling the dependence that makes the bound loose, and no usable treatment of that dependence is offered here.

8. Learning from execution outcomes

The learning objective is to estimate which configurations succeed for which tasks, and under what conditions. A first version can be a supervised classifier or regression model trained on a representative matrix of tasks and candidate configurations. Its inputs are the predictive dimensions R, A, V and U from Section 3.1, plus execution evidence from h for later-stage decisions. The label has to correspond to an independently assessed acceptance contract; a solver's own judgment of its output is a candidate feature at most.

8.1 Logging and label construction

Each trace records a pseudonymous task identifier, task-family and language labels, raw profile features, requirement and policy versions, candidate configurations, the selected action and its selection probability where randomised, model and tool versions, token and service usage, queue and execution latency, verification evidence, escalation reason, final outcome, and label source and time. Minimise or access-control sensitive content rather than copying it wholesale into training logs. The trace store inherits the retention and jurisdiction constraints of whatever it records.

Costs cover input and output processing, retries, retrieval, verification, orchestration and human handling. For self-hosted models, state amortised infrastructure and utilisation assumptions separately from marginal execution cost. The label distinguishes automated acceptance, successful referral, confirmed failure, unresolved status, and feedback that is simply missing. A referral can be operationally correct and still not count as an automatically solved task, which is why coverage appears as a constraint of its own in Equation (8).

8.2 Calibration and limited evidence

Train the quality predictor on training data; calibrate it on a separate split; tune routing thresholds without looking at the final test set. Reliability diagrams, Brier score and error within the accepted region each reveal something different about calibration. Methods such as temperature scaling want a compatible score model and empirical validation [5], and calibration on one workload establishes nothing about calibration after a domain shift [6]. Where the acceptance bound of Section 4.3 is to be claimed rather than estimated, the calibration split is the object conformal procedures consume, and its exchangeability with deployed traffic is the assumption under audit [10, 11].

A lower confidence bound can screen eligibility for task families with enough representative observations. It must not be dressed up as an exact guarantee for every arbitrary prompt. To illustrate: under independent Bernoulli trials, zero failures in 299 trials gives a one-sided 95 percent upper bound of about 1 percent on the failure probability — the rule-of-three approximation, exact value 0.997 percent. That inference is about the sampled population and its assumptions. Correlated documents, selective labels and repeated template instances each weaken it in practice.

8.3 Selection bias and controlled improvement

Execution logs show outcomes mainly for the actions that were selected. A router that always sends unfamiliar tasks to a frontier model cannot conclude from its own logs that a small model would have failed them. Safe shadow evaluations and randomised exploration on suitable workloads widen the coverage, and logged propensities make importance-weighted or doubly robust evaluation possible where overlap is adequate [8]. Extreme weights, sparse contexts and missing actions all need explicit diagnostics.

Test policy updates offline first, then in shadow mode, then on a limited monitored deployment. Exploration respects the same privacy and action constraints as ordinary execution. Online learning

operates under the same reliability thresholds as everything else, and under no circumstances may the router supply its own success labels. Keep a known-good policy and a rollback trigger for degradation in accepted error, referral load or latency.

8.4 The cost of keeping the registry current

The registry has to hold a measured success estimate for every task-family and configuration pair the router is permitted to use, and Section 8.2 gives the order of magnitude: roughly three hundred clean, independently audited observations to support a one percent failure claim for a single cell. I take six task families and twenty admissible configurations as a mid-sized portfolio, which is one hundred and twenty cells, and on the order of thirty-six thousand audited labels before the router may use its full action space. A smaller portfolio is the main lever anyone has on that number, which is why I return to portfolio size in Section 11.

And that cost recurs. A configuration is its model version, prompt, tools and context strategy taken together; change any one of them and the cell is invalid. Provider model versions turn over on a timescale of weeks to months, which can be shorter than the time it takes to re-certify against them. So the architecture is economical only if at least one of four conditions holds: the portfolio is small and deliberately stable; volume concentrates in a few task families, so that certification effort tracks traffic rather than the size of the cross-product; cells pool across families where transfer has been demonstrated rather than assumed; or version changes can be screened by a cheap regression suite that triggers full re-certification only when drift actually shows up.

This determines whether the router pays for itself, and unlike one-off labelling cost it does not amortise with volume. Certification and re-certification therefore belong in the same cost account as inference, and Section 10.3 lists them among the reported outcomes. Where the portfolio is large and versions turn over quickly, this term can exceed the inference it was introduced to save.

9. Analytical cost illustration

Every quantity in this section is either an assumption stated where it is used or a value derived from such assumptions. None of them is a vendor price, a benchmark measurement or an observed saving.

9.1 A two-stage cascade

Consider a two-stage cascade. The initial solver costs c_1 per task, its verifier v_1. A fraction e of tasks escalate to a stronger solver. Let o be routing overhead, profiling and any decomposition proposal included. Let c_2 be the cost of the direct strong-model baseline, averaged across the whole population.

Escalated tasks are selected for difficulty, and difficulty tracks the cost of handling it: harder requests run longer and consume more reasoning. Stage two on the escalated subset therefore costs $\kappa\, c_2$ for some $\kappa > 1$.[1] A naive comparison sets $\kappa = 1$ and biases the arithmetic toward the cascade, which is why every figure in this section carries κ explicitly. Equation (12) gives expected cascade cost on that accounting; Equation (13) the break-even condition that follows from it.

$$\mathbb{E}[C_{\text{cascade}}] = o + c_1 + v_1 + e\kappa c_2 \qquad (12)$$

$$\mathbb{E}[C_{\text{cascade}}] < c_2 \Leftrightarrow e < \frac{1}{\kappa}\left(1 - \frac{o + c_1 + v_1}{c_2}\right) \qquad (13)$$

[1] I keep κ scalar here for legibility. In a real workload it is a ratio of expectations over two different subpopulations, and it moves with output-length policy as much as with task difficulty, so anyone estimating it should report the escalated and unescalated token distributions and not only the ratio. The break-even algebra is what this section is for, and a scalar is enough to carry it.

I set o = 0.5, $c_1 = 2$ and $c_2 = 20$ in arbitrary cost units, with $v_1 = 1$ for the mid-range verifier labelled V1 in Table 4; the ten-to-one ratio between stages is what makes the cascade worth considering at all, and a narrower ratio closes the case quickly. First-stage fixed cost comes to 3.5 units. At $\kappa = 1$ the cascade is cheaper below e = 0.825. Push κ to 1.15 and break-even drops to 0.717; at 1.3, to 0.635; at 1.5, to 0.550. A thirty percent premium on escalated work removes roughly a quarter of the apparent operating range. The threshold concerns execution cost only, and establishes nothing about quality or latency. Everything that follows uses $\kappa = 1.3$.

9.2 Escalation fraction is not a free parameter

Presenting cost against a range of escalation fractions treats e as if it could be chosen on its own. It cannot. The escalation fraction, the error among accepted stage-one outputs, and the cost of verification all follow from a single design decision: which verifier is built. A table that varies e while implicitly holding quality and verification cost fixed is describing operating points that do not exist.

Parameterise the verifier by its operating characteristics instead, as in Equation (14). Let f_1 be the raw failure rate of stage one on the population, ρ the fraction of genuine failures the verifier catches, and φ the fraction of correct outputs it flags unnecessarily.

$$e = \rho f_1 + \varphi(1 - f_1), \qquad r_1 = \frac{(1-\rho) f_1}{1 - e} \qquad (14)$$

The escalated subset is a mixture whose composition shifts with the verifier, so stage-two error cannot be held constant across operating points either. It holds ρf_1 genuinely failed tasks, on which stage two has error r_h, and $\varphi(1 - f_1)$ correct tasks flagged in error, on which stage two has the lower error r_e. Equation (15) gives the resulting rate on the escalated subset.

$$r_2 = \frac{\rho f_1 r_h + \varphi(1 - f_1) r_e}{e} \qquad (15)$$

$$R = (1 - e) r_1 + e\, r_2 = (1 - \rho) f_1 + \rho f_1 r_h + \varphi(1 - f_1) r_e \qquad (16)$$

The closed form on the right of Equation (16) pulls apart the three routes to a wrong final answer, and each one belongs to a different part of the design. First, failures the verifier passed, governed by recall alone. Second, failures correctly escalated on which stage two fails anyway, governed by the strong model. Third, correct work escalated unnecessarily and then broken by stage two — the price of over-flagging. So verifier recall puts a floor under achievable quality that no amount of stage-two capability lifts, and over-flagging turns out to be actively harmful rather than merely wasteful.

Those same parameters fix the baseline. Run stage two on every task and it handles f_1 hard tasks and $1 - f_1$ easy ones, so the direct strong-model baseline has error $f_1 r_h + (1 - f_1) r_e$. With $f_1 = 0.20$, $r_h = 0.15$ and $r_e = 0.02$, that comes to 0.046, derived rather than assumed. Table 4 sets five verifier designs against it, each carrying the verification cost such a design would plausibly incur. V0 checks schema and format only. V1 adds executable tests. V2 adds independent model review. V3 is V2 tuned to flag broadly. V4 adds a human spot-check on every output.

Table 4. Cost and quality by verifier design, at κ = 1.3, against a direct strong-model baseline costing 20 units with error 0.046

Verifier	Recall ρ	False flag φ	Verify cost v_1	Escalation e	Accepted error r_1	Total cost (vs 20)	Final error R (vs 0.046)
V0	0.50	0.05	0.3	0.140	0.116	6.44 (67.8% lower)	0.116 (152% worse)

Verifier	Recall ρ	False flag φ	Verify cost v_1	Escalation e	Accepted error r_1	Total cost (vs 20)	Final error R (vs 0.046)
V1	0.80	0.05	1.0	0.200	0.050	8.70 (56.5% lower)	0.065 (41% worse)
V2	0.95	0.05	2.5	0.230	0.013	10.98 (45.1% lower)	0.039 (15% better)
V3	0.95	0.20	3.0	0.350	0.015	14.60 (27.0% lower)	0.042 (9% better)
V4	0.99	0.40	6.0	0.518	0.004	21.97 (9.8% higher)	0.038 (17% better)

The viable window is narrow, and it closes at both ends. V0 is the cheapest configuration in the table and by a wide margin the worst. A cost comparison that dropped the quality column would pick it. V1 catches four failures in five and still fails non-inferiority. V2 is the first design that both saves money and improves on the baseline, and it manages that because verification catches failures the direct strong-model baseline never examines. V3 passes on quality but spends a third of the remaining savings escalating work that did not need it, and ends up worse than V2 on quality despite identical recall — the third term of Equation (16), made visible.

V4 is the instructive failure. Best recall, lowest accepted error, good final quality, and it costs more than not cascading at all. Verification thorough enough to be right nearly always consumes the savings that motivated the cascade, and at that point the architecture should be abandoned in favour of routing everything to the strong model. A cascade is therefore justified only over a bounded range of verifier quality: weak enough to be cheap, strong enough to be safe. Those two conditions do not always overlap. Whether they overlap on a given workload is an empirical question about f_1, κ and the attainable (ρ, φ, v_1) frontier, and it is the first thing an evaluation should settle.

The figures depend on the assumed values, but the qualitative conclusions are not all equally robust. That a weak verifier yields the cheapest and worst configuration follows from Equation (16) for any $r_h > r_e$ and does not depend on the particular numbers at all. That the window closes at V4 depends on the assumed verification costs, which are the least defensible assumption in the section, since a cheaper route to high recall would reopen it. Measuring the cost of verification at each attainable recall is accordingly the measurement on which this architecture stands or falls.

9.3 Reliability and latency conditions

Equation (16) assumes every task finishes at one of the two stages with no referrals, and that the tasks stage one fails are the ones stage two also finds harder. Both are modelling assumptions rather than observations. On a workload where stage-one and stage-two difficulty are unrelated, r_h would equal r_e and most of the cascade's quality advantage would disappear.

A cost comparison is meaningful only when both policies meet the same success criteria and the same residual-risk constraints. The comparison is therefore reported as a cost-quality frontier and a risk-coverage curve; a single cost figure conceals the trade-off Table 4 sets out.

Under serial execution, escalation adds the first-stage delay before the stronger solver even starts. Average latency can improve while the slowest requests get slower, so keep the direct strong-model baseline available for cases where the expected extra attempt would breach the deadline. Verification

cost is also verification latency: the V4 design that fails on cost fails a tight deadline for the same reason. Routing decisions therefore need measured tail latency and queue behaviour at the concurrency the deployment actually runs at, which is rarely the concurrency a benchmark measures.

10. Proposed empirical evaluation

The central hypothesis is that evidence-based routing lowers total completion cost at a prespecified quality level, relative to sending every request to the strongest general model available. Two more follow it: that dynamic evidence improves decisions over prompt-only routing, and that validated decomposition lowers cost for separable workflows. All three can fail: on uniformly difficult workloads, with weak verifiers, or where coordination overhead dominates. Section 9.2 names verifier recall as the parameter most likely to decide the outcome.

10.1 Workload and reference outcomes

Build a preregistered, stratified corpus covering deterministic transformations, structured extraction, grounded question answering, cross-document synthesis, executable coding and multi-stage planning. Include the awkward cases deliberately: ambiguous requests, long but repetitive inputs, short difficult ones, degraded document quality, minority languages, unavailable tools, policy-constrained work. Use rights-cleared public tasks and consented or de-identified enterprise samples, and publish the access and licensing conditions.

Partition by document family, source organisation, template and, where it matters, time, so that near-duplicates cannot straddle training and test. Keep training, calibration, policy-validation and held-out test splits separate. The calibration split is the one the acceptance procedure of Section 4.3 consumes, and it must not be reused for threshold tuning. Evaluate out-of-distribution sets after policy selection, not before. Specify candidate model and tool snapshots, context budgets, decoding settings, hardware, concurrency and source-access rules for every run.

Use exact reference outcomes where they exist and independent expert rubrics where they do not. Keep reviewers blind to model identity and routing condition. Define critical errors before evaluation rather than after, measure reviewer agreement, and adjudicate the disputed cases. Collect multiple runs per task wherever stochastic variation matters. Determine sample size from a prespecified non-inferiority margin, the target power, the expected failure rate and the degree of clustering. With source-clustered tasks the effective sample is well below the nominal one, and that is the term most often dropped.

10.2 Baselines and ablations

Compare a direct strong-model baseline, a direct small-model baseline, deterministic rules with fallback, a prompt-only learned router, a fixed ordered cascade, and the full adaptive policy. Include comparable implementations of established routing or cascade methods [2, 3] and of conformal cascading [12, 14] wherever their assumptions fit. Give every system equivalent tool access, data constraints, final acceptance contracts and accounting boundaries. Strong-model runs get the same tuning effort as the router arms; a weakly prompted baseline invalidates the comparison.

I make one arm mandatory rather than optional: a non-routing efficiency baseline, which is exact and semantic caching with template detection and deduplication, applied to the direct strong-model baseline. Repeated and near-duplicate requests are common in enterprise workloads, and caching captures that saving for a small fraction of what the router costs to design, certify and maintain. A routing result not measured against this arm cannot be attributed to routing, since the cheaper alternative was never given the chance to produce the same savings.

Ablate one at a time: deterministic resolution, dynamic execution features, independent verification, calibration, decomposition, learned updates. Separate the effect of better routing from the effect of additional inference or verifier computation, since Section 9.2 shows how easily the two get confused. Compare against a hindsight portfolio oracle only as an explicitly unattainable reference, since it uses outcome information no real router has.

10.3 Decision criteria

The primary outcome is total cost per successfully completed task, under quality non-inferiority to the baseline, reported alongside automated coverage and independently audited accepted error. Report also: total spend per submitted task, human referral and failure rates, p50/p95/p99 completion latency, deadline misses, escalation depth, calibration error, verifier false acceptance and false rejection, and domain-specific critical errors. Include abandoned and failed work in all of these; excluding it is the most common route to a favourable-looking cost per successful task.

Certification and re-certification effort enter the same cost accounting as inference, on the argument of Section 8.4. Two results that differ on whether this term is included are not comparable, so the accounting boundary has to be stated alongside the headline figure.

Use paired comparisons on the same tasks, with cluster-aware confidence intervals wherever observations share sources or workflow dependencies. Preregister the non-inferiority margin. Report uncertainty on cost as well as on quality. A positive result needs savings that survive all overhead, an adequate bound on quality loss, and compliance with the group-conditional constraints of Equation (9) in every prespecified group. Publish the negative and subgroup results alongside the aggregate.

11. Implementation and operational boundaries

A working prototype needs a task-contract service, a capability registry, a profiler, a policy engine, an executor, a verifier and a trace store, with interfaces stable enough that these can change independently. The executor gets least-privilege tool access, bounded timeouts, idempotency controls for retryable actions, and a hard separation between producing a recommendation and carrying out an external change. Escalating to a more capable solver must not enlarge the permissions available to it: the permission set is fixed by the task contract and the caller's role, and not by the configuration that happens to be chosen.

An initial policy can pair rules for certified deterministic tasks with a calibrated predictor over a small set of configurations. Larger portfolios raise the evaluation burden, thin out the actions, and cost more to version-manage, in the manner Section 8.4 quantifies. Add candidates only where they show useful specialisation or better cost-quality behaviour on the target workload. The architecture should justify its own complexity by measured benefit, and Section 10.3 states the form that evidence would take.

11.1 Clarification can be the cheapest useful action

Take the request "convert 7 cups to mL". Computationally trivial once the cup convention is fixed, and ambiguous until it is, since US customary, US legal, metric and imperial cups differ enough that the answer spans roughly 1,656 to 1,989 mL.[2] A larger model does not resolve that. Only the requester, or a stated default, does. The example is small on purpose and its value is narrow: it shows why determinism

[2] The four conventions are the US customary cup (236.588 mL), the US legal cup (240 mL), the metric cup (250 mL) and the imperial cup (284.131 mL); the range in the text is seven times the smallest and seven times the largest. I chose the example because its ambiguity is documented and uncontroversial. Most ambiguity encountered in practice is neither, which is why the profiler has to treat A as an estimate rather than a lookup.

and ambiguity have to be separate signals, because this task is fully determinate and wholly ambiguous at the same time.

The consequential cases look quite different, and they are what the architecture is for. A banking migration plan can be internally coherent, technically defensible, and still inadmissible, because what binds is authorisation rather than capability. Its profile is dominated by S and T — irreversible actions, privileged systems, a named approver — while R is unremarkable. No increase in model capability changes the admissible set, and a router scoring only difficulty will route such a request confidently and wrongly. This is where a scalar complexity index fails most clearly, and why Section 3.1 keeps the constraint and floor dimensions apart from the predictive ones.

A contract comparison can fail for a duller reason: an attachment is missing, and the correct next action is retrieval or a request for the document. Human involvement brings institutional responsibility and context. It does not bring a blanket guarantee of higher accuracy, which is why Section 5.1 scores referral against measured reviewer performance. Measure human workload, disagreement and turnaround alongside the automated numbers.

11.2 Threat model and abuse resistance

Retrieved documents and tool outputs are data. They are not authority to alter the router's policy. An instruction embedded in a document telling the system to declare a task low risk, skip a verifier, or take an expensive route must not override the acceptance contract. Resource quotas and escalation limits contain attempts to consume excessive inference, and keeping provenance for the evidence behind any routing change lets a reviewer tell a real task requirement from manipulated input.

Injection is not the primary exposure, though. The leverage sits earlier. The request determines S; S determines the reliability floor, the mandatory oversight, and the admissible set. Let a caller declare their own stakes tier and that caller can choose their own cheap route and skip the review their task required. No injection is needed, and nothing in the resulting trace appears anomalous. Stakes must therefore be derived from attributes the caller does not control: the authenticated caller's role, the data classes the request touches, the systems the requested action would write to, and how reversible those writes are. Where request text is the only signal available, the tier it implies is a lower bound for policy to raise, never a ceiling. The acceptance contract inherits the same asymmetry — a caller may tighten it, and may not loosen it below the floor their role and data class impose.

Provider outages, rate limits, slow queues and model changes all move the cheapest feasible path around. A fallback is usable only if it satisfies the same constraints; availability is not admissibility. Cache reuse has to respect user permissions, input identity, source freshness, model configuration and acceptance-contract version, because a query that looks identical can require a different answer once the underlying records have changed.

11.3 Deployment sequence

Start with offline evaluation and independently reviewed labels. Move to shadow decisions on live tasks. Then introduce automated routing in a bounded, reversible workload, and expand coverage only once measured performance meets the contract. Freeze policy and model versions for comparison windows, keep audit sampling running across accepted cases, and trigger rollback or more conservative routing when drift invalidates the calibration that Section 4.3 depends on.

An operational dashboard should carry cost per successful outcome, automated coverage, accepted error, human backlog, deadline compliance, and performance broken out by task family and language. Token

counts and large-model avoidance are diagnostics, not objectives. Avoiding an expensive model is a failure whenever the downstream cost of the resulting mistakes exceeds the inference it saved.

12. Discussion, limitations, and conclusion

12.1 What the architecture does and does not establish

The proposal treats intelligence allocation as a systems problem, spanning interpretation, computation, generation, evidence and responsibility. Its advantage, if it has one, arises where solver capabilities are complementary and the router can predict those differences cheaply enough to be worth predicting. Nothing in it implies that model size measures intelligence, that every task has a stable complexity score, or that more capability always means more correctness.

The strongest prospective contribution is a connection made explicit: between task contracts and resource allocation. Rather than buying maximum capability for every request, an organisation states the outcome it needs and then tests which execution policy delivers it. A side benefit is that ordinary algorithms stay first-class components of an AI application instead of being displaced on principle. Whatever value the router has comes from skipping inference that was not needed, while noticing when more capability, more evidence, or human judgment is in fact required.

12.2 Limitations

An earlier version of this design called reliable guarantees beyond its reach. That was too strong, and correcting it narrows the claim rather than widening it. Marginal, distribution-free guarantees on expected loss over a calibration population are available, and they are the intended basis for the acceptance rule in Section 4.3 [10, 11, 12]. What stays out of reach is a per-request guarantee for an arbitrary novel prompt. The available results are marginal over a population assumed exchangeable with the calibration set, and both that population and the exchangeability assumption are exactly what a deployed router disturbs when it changes its own routing. A guarantee of this kind is therefore something a deployment maintains and monitors, under assumptions its own routing can invalidate; Section 11.3 makes drift detection and rollback a condition of operating at all.

No empirical validation appears in this paper: not of the complexity vector, the routing predictor, the decomposition policy, or the quality thresholds. The role partition in Section 3.1 makes each dimension testable. It does not establish that these are the right dimensions, that they separate in practice, or that annotating them is reliable. Richer features may improve predictions while adding cost, latency and privacy exposure.

Ground truth is hard for open-ended analysis, for strategy, and for consequential decisions generally. A deliverable can satisfy a rubric and still fail in use, and the feedback may not arrive until after the model version has changed. Common training data and shared source errors correlate failures across solvers and verifiers, which is the failure mode Section 6.1 flags as most damaging precisely because it breaks the acceptance bound without any noise. Human review varies as well. What these have in common is that they are measurement problems: they yield to better reference outcomes and independent audit, and a threshold setting cannot touch them.

The analytical illustration assumes a two-stage serial cascade and a single escalation premium κ that a real workload would have to estimate rather than assume. Its verification costs are the weakest assumption it makes: the five designs in Table 4 carry plausible but invented values for v_1, and since Section 9.2 shows the viable window is closed from above by exactly that quantity, a cheaper route to high recall would change the conclusion. The assumption that stage-one and stage-two difficulty

coincide carries as much weight and has been tested as little. Real deployments add batching, caching, concurrency, retries, variable output lengths, queueing, and heterogeneous human costs. The decomposition procedure is restricted to depth one, and it offers no account of when deeper structure would repay its planning cost.

Finally: a literature review cannot establish exhaustive novelty, and certainly not patentability. The related work here is explicitly partial. Adjacent literatures on mixture-of-experts routing, speculative decoding and early-exit inference all address the same underlying question — allocate less computation to easier inputs — and none of them is treated. The contribution is an integration and a research agenda, with the evaluation obligations set out in Section 10 rather than deferred to later work.

12.3 Conclusion

Automatic model selection should select a complete path to a verified outcome: tools, evidence, retries and oversight included. The Adaptive Intelligence Router operationalises that view through a task profile whose dimensions carry distinct and stated roles, a constrained solver portfolio, an acceptance rule calibrated on post-verification evidence rather than pre-execution prediction, and a bounded cycle of selection, execution, verification and revision. Decomposition carries the same allocation down from whole requests to individual workflow components, and audited feedback is what makes measured policy improvement possible.

Proportionate allocation is the governing policy. Do not invoke learned inference where ordinary computation meets the task contract. Do not invoke a more expensive configuration where a validated cheaper one suffices. And do not withhold stronger capability or human judgment where the evidence calls for it. Whether any of this yields an engineering advantage is an empirical question, and Section 9.2 suggests where it gets decided: not in the choice of models, but in the quality and cost of the verifier, and in whether the registry can be kept current for less than the inference it saves. The benchmark, cost accounting and acceptance criteria proposed here are a concrete way to find out.

References

1. Chen, L., Zaharia, M., & Zou, J. (2023). FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. arXiv:2305.05176.
2. Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W., & Stoica, I. (2024). RouteLLM: Learning to Route LLMs with Preference Data. arXiv:2406.18665.
3. Dekoninck, J., Baader, M., & Vechev, M. (2025). A Unified Approach to Routing and Cascading for LLMs. Proceedings of the 42nd International Conference on Machine Learning, PMLR 267:12987–13010. arXiv:2410.10347.
4. Geifman, Y., & El-Yaniv, R. (2019). SelectiveNet: A Deep Neural Network with an Integrated Reject Option. Proceedings of the 36th International Conference on Machine Learning, PMLR 97:2151–2159. proceedings.mlr.press/v97/geifman19a.
5. Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). On Calibration of Modern Neural Networks. Proceedings of the 34th International Conference on Machine Learning, PMLR 70:1321–1330. proceedings.mlr.press/v70/guo17a.
6. Kadavath, S., et al. (2022). Language Models (Mostly) Know What They Know. arXiv:2207.05221.
7. Zheng, L., et al. (2023). Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. Advances in Neural Information Processing Systems 36. arXiv:2306.05685.

8. Dudík, M., Langford, J., & Li, L. (2011). Doubly Robust Policy Evaluation and Learning. Proceedings of the 28th International Conference on Machine Learning. arXiv:1103.4601.
9. Moslem, Y., & Kelleher, J. D. (2026). Dynamic Model Routing and Cascading for Efficient LLM Inference: A Survey. Transactions on Machine Learning Research. arXiv:2603.04445, version 3.
10. Angelopoulos, A. N., Bates, S., Candès, E. J., Jordan, M. I., & Lei, L. (2021). Learn then Test: Calibrating Predictive Algorithms to Achieve Risk Control. arXiv:2110.01052.
11. Angelopoulos, A. N., Bates, S., Fisch, A., Lei, L., & Schuster, T. (2022). Conformal Risk Control. arXiv:2208.02814.
12. Dou, Y., Lian, S., & Li, S. (2026). Conformal Cascade: Distribution-Free Accuracy Guarantees for Multi-Tier LLM Inference. arXiv:2607.25018.
13. Kotte, V. (2026). PASC: Pipeline-Aware Conformal Prediction with Joint Coverage Guarantees for Multi-Stage NLP and LLM Pipelines. arXiv:2605.18812.
14. Guo, D., Wu, J., & Yiu, S. M. (2026). RouteNLP: Closed-Loop LLM Routing with Conformal Cascading and Distillation Co-Optimization. arXiv:2604.23577.
15. Chow, C. K. (1970). On Optimum Recognition Error and Reject Tradeoff. IEEE Transactions on Information Theory, 16(1), 41–46. doi:10.1109/TIT.1970.1054406.
16. Mozannar, H., & Sontag, D. (2020). Consistent Estimators for Learning to Defer to an Expert. Proceedings of the 37th International Conference on Machine Learning, PMLR 119:7076–7087. arXiv:2006.01862.
17. Verma, R., & Nalisnick, E. (2022). Calibrated Learning to Defer with One-vs-All Classifiers. Proceedings of the 39th International Conference on Machine Learning. arXiv:2202.03673.
18. Altman, E. (1999). Constrained Markov Decision Processes. Chapman & Hall/CRC, Stochastic Modeling Series. ISBN 978-0849303821.