

APUS-OpenJev- v1: Learning to Decide Across Compute Budgets

gumpcheng · zhangxu | APUS AI-LAB

Technical Report v1.1 · September 2026

CONTENTS

APUS-OpenJev-v1: Learning to Decide Across Compute Budgets	
Abstract	
Model value	
1. From Text Completion to Language-Conditioned Decisions	
2. A Shared Model Across Compute Budgets	
3. Learning Decisions and Execution Depth Together	
3.1 Joint supervision and cross-depth knowledge transfer	
3.2 Text co-training for workflow continuity	
3.3 A staged 4B path and an independent 9B study	
4. A Curriculum Organized Around Decision Capabilities	
5. Jet-DCRL: Decision Utility and Probability Quality	
6. Performance: Where the Savings Can Come From	
7. Initial Evidence and the Evaluation Agenda	
7.1 Decision accuracy	
7.2 Response latency	
7.3 Evaluation agenda	
Acknowledgments	

APUS-OpenJev-v1: Learning to Decide Across Compute Budgets

Technical Report v1.1

gumpcheng (<https://huggingface.co/xDAN2099>) · zhangxu
[APUS AI-LAB](#)

Abstract

Many useful language-model interactions end in a decision: choosing a browser action, applying a principle, routing a workflow, or determining whether evidence supports an attribute. These interactions require language understanding, but their outputs often occupy a small, explicitly defined space. APUS-OpenJev-v1 organizes model computation around that distinction. It combines language-conditioned candidate decisions, jointly trained execution depths, and a text objective for workflow steps that require open responses. A shared model exposes low and high effort modes, connecting application budgets to the depth of neural computation. Cross-depth self-distillation transfers information from the deeper decision distribution into an earlier exit. We describe the delivered training framework, its 4B and 9B instantiations, and Jet-DCRL, a separately implemented research extension for decision utility and probability quality. Development results establish an initial accuracy reference; the evaluation agenda targets the broader relationship between decision quality, execution cost, and workflow outcomes.

MODEL VALUE

APUS-OpenJev brings language understanding to the decision points where applications need an actionable choice. Its value comes from combining decision quality, a direct output path, and control over computational effort within one deployable model family.

- **Evidence-informed decisions.** Tasks, policies, and candidate actions are expressed in natural language, so a shared model can address changing business rules and action spaces. The 9B model achieves 85.00% accuracy on the development comparison panel, alongside 82.50% for the Jev API. Section 7 presents the shared evaluation setting and results.
- **Less work between understanding and action.** Candidate scoring produces a bounded decision without decoding a complete structured document. This targets the repeated selection steps found in browser agents, business routing, and workflow handoffs.

- **Compute that fits the workflow.** Jointly trained low and high effort modes let applications choose the execution budget appropriate to each decision point. This creates a practical basis for balancing decision quality with latency and resource cost.
- **Deployment under application control.** Downloadable 4B and 9B weights and an accompanying runtime support local decision execution, giving teams control over infrastructure, data handling, and integration into existing systems.

For browser agents, the target is choosing the next action from page evidence and a goal. For high-frequency business services, it is applying a supplied rule or routing a request to an appropriate next step. For workflow handoffs, it is returning a decision that host code can map into a structured action. These scenarios share the same language-conditioned decision interface; their production value can be evaluated through both decision correctness and completed workflow outcomes.

1. From Text Completion to Language-Conditioned Decisions

The value of a decision model lies in interpreting a changing problem, rather than memorizing a fixed business taxonomy. A browser page presents different elements at every step. A workflow may introduce a new policy, tool, or routing destination. The relevant candidate set therefore belongs to the request itself. APUS-OpenJev-v1 receives the context, question or instruction, and candidate descriptions in language, then returns a decision over that request-specific set.

This representation preserves a central strength of language models: task meaning is communicated through the input. A common model can process evidence-based questions, natural-language principles, workflow states, and browser actions without requiring a separate output classifier for every business category. Compact output labels identify candidates; their meaning comes from the descriptions supplied for the current request. They are an interface to the decision space, not permanent class identities.

For candidates $A(x) = \{a_1, \dots, a_k\}$, an execution depth d produces logits $z_d(x, A)$. The decision distribution is:

$$p_d(a_i \mid x, A) = \exp(z_d, i) / \sum_j \exp(z_d, j).$$

Normalization takes place within the legal candidate set. A deterministic application layer can map the selected label into its structured response. This removes the need to generate JSON punctuation or repeatedly decode a known answer schema. It guarantees membership in the supplied candidate set, while semantic correctness remains a learned capability that must be evaluated. Open text is retained where the task genuinely requires it, such as entering a value during a browser workflow.

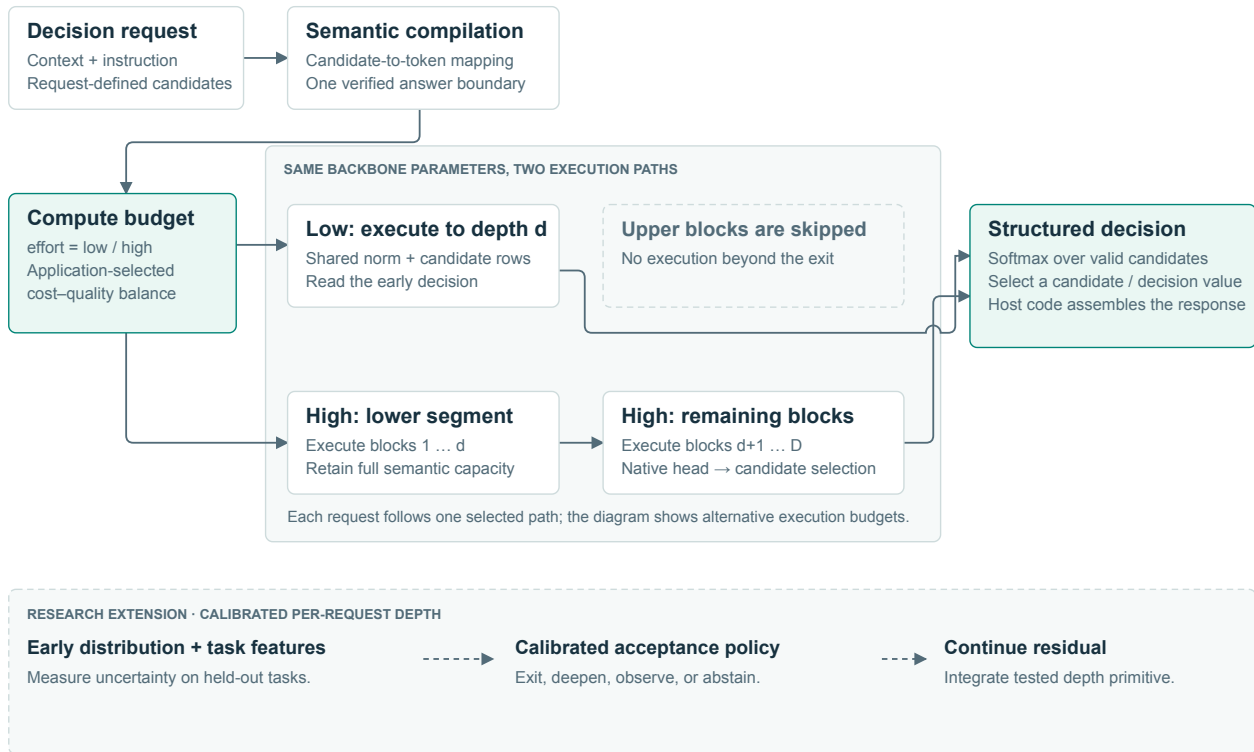
2. A Shared Model Across Compute Budgets

APUS-OpenJev-v1 makes execution depth an explicit part of the decision framework. The same parameter set supports an earlier decision exit and a deeper exit. Applications select `effort="low"` or `effort="high"` according to the latency and quality needs of a workflow. The two modes change the neural computation performed, rather than merely changing a response-length limit.

APUS-OPENJEV · ARCHITECTURE STUDY

Spend computation where the application needs it.

One model family · selectable execution depth · bounded decisions without a text-decoding loop



Solid flow: release execution paths Dashed region: proposed serving integration Text generation is a separate path.

Figure 1. Shared language representations support decisions at different execution depths. Joint training connects the exits through supervision and distributional knowledge transfer; deployment exposes the resulting compute-budget choices.

The shared prefix builds a contextual representation of the instruction, evidence, and candidate semantics. An earlier readout extracts a candidate decision from that representation. The deeper path applies additional backbone transformations before producing its decision. Both exits use the model's language output space, preserving a common interpretation of candidate labels across depths. This avoids introducing a collection of task-specific business heads as the organizing architecture.

Depth-sensitive execution also requires a careful separation between the representation used for a decision and the state used for further computation. In the depth executor, normalization for a readout does not overwrite the residual state required by subsequent blocks. This distinction makes a shallow decision and a deeper continuation conceptually compatible. The current release establishes the selectable exit mechanisms; learned per-request routing and production continuation reuse remain separate evaluation targets.

The important training question is whether an intermediate representation has acquired enough decision structure to be useful. Merely exposing an earlier layer does not answer it. Our approach therefore supervises the exits jointly and couples their candidate distributions. Lower-cost execution becomes a training objective with measurable quality, rather than an inference shortcut assumed to preserve behavior.

3. Learning Decisions and Execution Depth Together

3.1 JOINT SUPERVISION AND CROSS-DEPTH KNOWLEDGE TRANSFER

For a supervised decision with target distribution q , the delivered objective is:

$$L_{\text{decision}} = 0.5 \text{ CE}(q, p_{\text{low}}) + 0.5 \text{ CE}(q, p_{\text{high}}) \\ + 0.1 \text{ KL}(\text{stopgrad}(p_{\text{high}}) || p_{\text{low}}).$$

Both exits receive direct task supervision. The deeper exit additionally acts as a distributional teacher for the earlier exit. Its detached probabilities convey relative candidate preferences beyond the identity of the correct answer, while stop-gradient keeps the distillation target from being changed through that term. The deeper pathway continues to learn through its own supervised objective.

This arrangement is an internal transfer mechanism: the model learns to express useful decision structure at more than one computational budget. It is especially relevant when several candidates are plausible and the earlier exit must preserve their distinctions with fewer transformations. Its effect can be measured through the quality–cost curve of the two exits.

The supervision and transfer terms operate together during each decision update, enabling the shared model to learn both execution budgets in one coordinated optimization process.

3.2 TEXT CO-TRAINING FOR WORKFLOW CONTINUITY

Some decisions lead directly to an open response: a search query, a form value, or another textual argument. TYPE examples therefore use the full-depth native text cross-entropy objective. Decision and text examples share the model while following the objective appropriate to their output space.

The resulting training interface separates bounded selection from open completion without discarding either capability. A browser-oriented system can learn which action to take and retain a path for supplying the text that action requires. This is a training design for workflow continuity; full end-to-end browser competence requires additional execution-based evaluation beyond static action selection.

3.3 A STAGED 4B PATH AND AN INDEPENDENT 9B STUDY

The 4B model follows a concrete staged adaptation path. A full-depth starter first learns from a focused mixture of decision and TYPE examples. The resulting parameters initialize a broader run that jointly optimizes the two decision exits and the text objective. The initial stage establishes task-facing behavior; the subsequent stage introduces the shared compute-budget learning objective across a wider task mixture.

The 9B study starts independently from its own base initialization and uses the same registered curriculum and joint objective. It does not inherit the 4B starter. This provides a model-scale comparison with a common main training protocol while preserving the distinction in initialization history. The primary 9B result reported here uses an intermediate training version; the full-schedule version is also retained.

APUS-OPENJEV · ARCHITECTURE STUDY

Learning decisions across compute budgets.

Candidate semantics enter as language; depth becomes a learned decision interface.

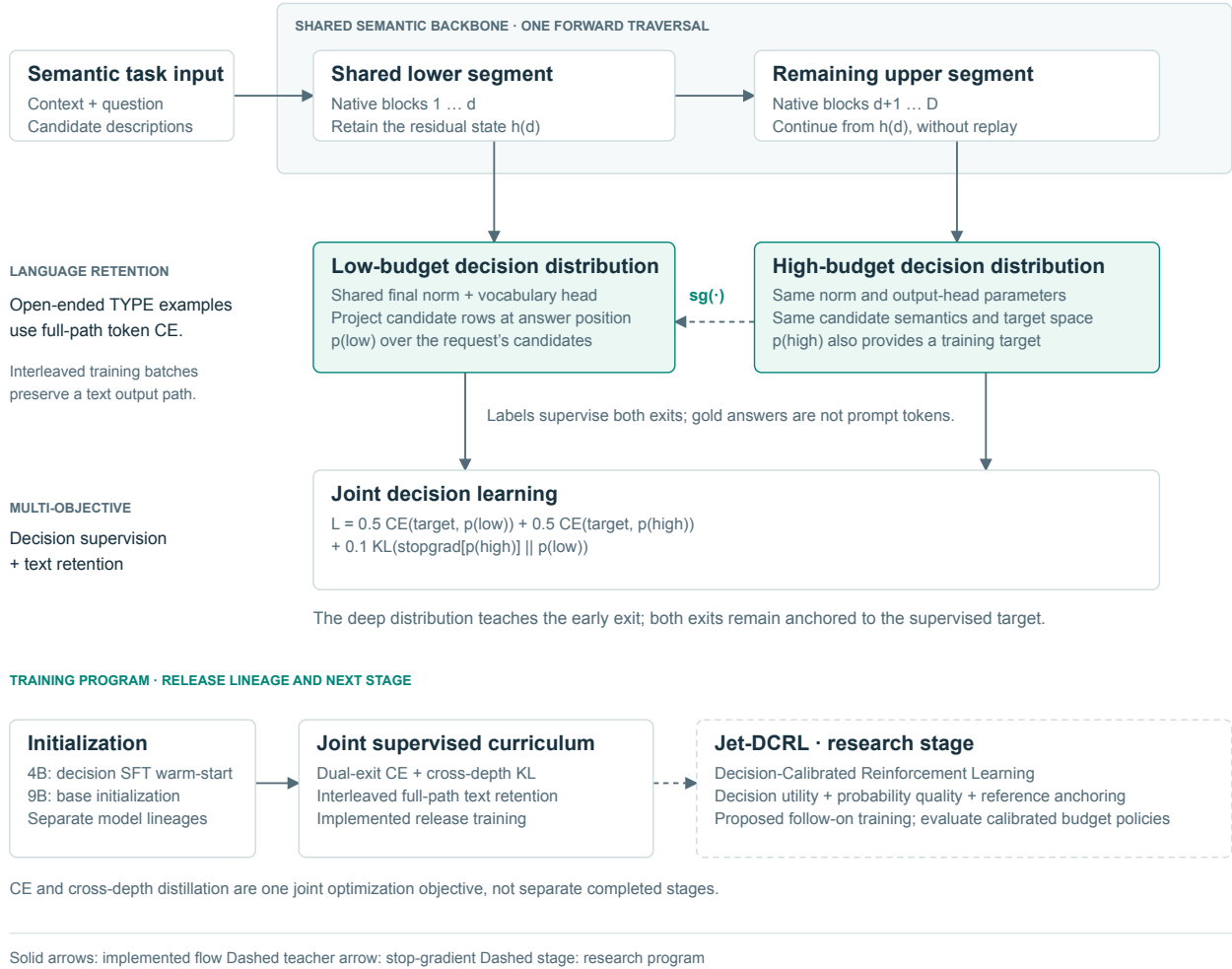


Figure 2. The delivered framework combines task alignment, mixed decision/text supervision, and joint depth learning. Jet-DCRL extends the research program toward candidate utility and probability quality; it is not part of the published weights described here.

4. A Curriculum Organized Around Decision Capabilities

The main curriculum allocates **97.34%** of training records to candidate decisions and **2.66%** to open-text TYPE examples. The source shares below describe the registered training mixture. Related examples are tracked by parent episode, dialogue, or source context to preserve their provenance.

Training source	Share	Capability emphasized
Mind2Web browser Choice	30.22%	Selecting actions from page evidence and goals
Mind2Web browser TYPE	2.66%	Producing necessary workflow text
HelpSteer3 principle	21.85%	Applying a stated natural-language principle
Schema-Guided Dialogue	8.62%	Decisions conditioned on dialogue and workflow state
GoEmotions attribute Score	8.40%	Judging whether a specified attribute applies
BoolQ	10.09%	Evidence-grounded yes/no decisions
MNLI	16.81%	Entailment, contradiction, and neutral relations
Local counterfactual records	1.34%	Sensitivity to decision-relevant factual changes

Source percentages are calculated over the main training schedule and rounded to two decimal places. Their displayed sum may differ slightly from 100% because of rounding.

The common candidate interface makes these tasks compatible without erasing their differences. Browser examples ground choices in visible elements and goals; principle examples require conditioning on the rule currently provided; entailment and binary questions emphasize relations between claims and evidence. Counterfactual records target changes that should alter a decision, rather than encouraging invariance to every input perturbation.

Data conversion, candidate construction, parent grouping, and training manifests preserve the relationship between source evidence and target decisions. The main schedule is a heterogeneous task mixture; starter records and main records are tracked separately, including potential overlap.

5. Jet-DCRL: Decision Utility and Probability Quality

A useful next step is to distinguish three properties that accuracy alone cannot express: whether a decision is correct, whether its probability reflects uncertainty, and whether its consequence is valuable under the application’s costs. We call this research extension **Jet-DCRL — Decision-Calibrated Reinforcement Learning**.

Its candidate-space objective combines proper-scoring components, reference preservation, and decision utility:

$$L_{\text{DCRL}} = \text{mean}_d [\text{NLL}(q, p_d) + \lambda_B \text{Brier}(q, p_d)]$$

$$+ \lambda_{\text{ref}} \text{KL}(p_{\text{high}} || p_{\text{reference}})$$

$$+ \lambda_U L_{\text{utility}}(p_{\text{low}}) .$$

The NLL and Brier terms supervise probabilities at both exits. The reference term anchors the deeper distribution to a frozen reference. The utility term targets the earlier exit, where better decisions have particular value for low-cost execution. Candidate utilities are supplied as verified training information; they are not inferred merely from the model’s confidence.

For bounded candidate sets, the exact utility objective is tractable:

$$L_{\text{utility}} = -\sum_a p_{\text{low}}(a) u(a) .$$

A sampled alternative draws actions from the current shallow policy and uses an expected-utility baseline in its policy-gradient estimator. Comparing exact and sampled optimization is valuable here: when the action space is small, sampling should earn its complexity through evidence rather than being assumed necessary.

The research objective and supporting implementation are separate from the released models: **the published weights in this report have not been trained with Jet-DCRL**. Its purpose is to test whether utility-aware learning can improve economical decisions while preserving probability quality and deeper-path behavior. Independent NLL, Brier, reliability, and risk–coverage measurements will assess the resulting probabilities.

6. Performance: Where the Savings Can Come From

The framework offers complementary opportunities to reduce decision cost. First, a bounded decision can be obtained from candidate scores without autoregressively generating an entire structured document. Second, an earlier exit executes fewer backbone blocks. Third, selected candidate projections can reduce output-head work where the execution path supports them. These mechanisms address different components of inference and should be measured separately before their gains are combined.

The cost model includes context processing, executed depth, output projection, and any additional queries sharing the context. Shared-context execution offers a further opportunity to amortize input processing across related decisions. Its design must preserve both attention caches and the backbone’s recurrent state, with numerical equivalence established before performance measurements.

Our performance target is the operating curve between useful decisions and deployment cost: end-to-end latency, sustainable request rate, memory demand, and task success under a declared workload. Local model-forward timings and remote HTTP response times are reported separately; the next deployment comparison will align request boundaries and workloads.

7. Initial Evidence and the Evaluation Agenda

7.1 DECISION ACCURACY

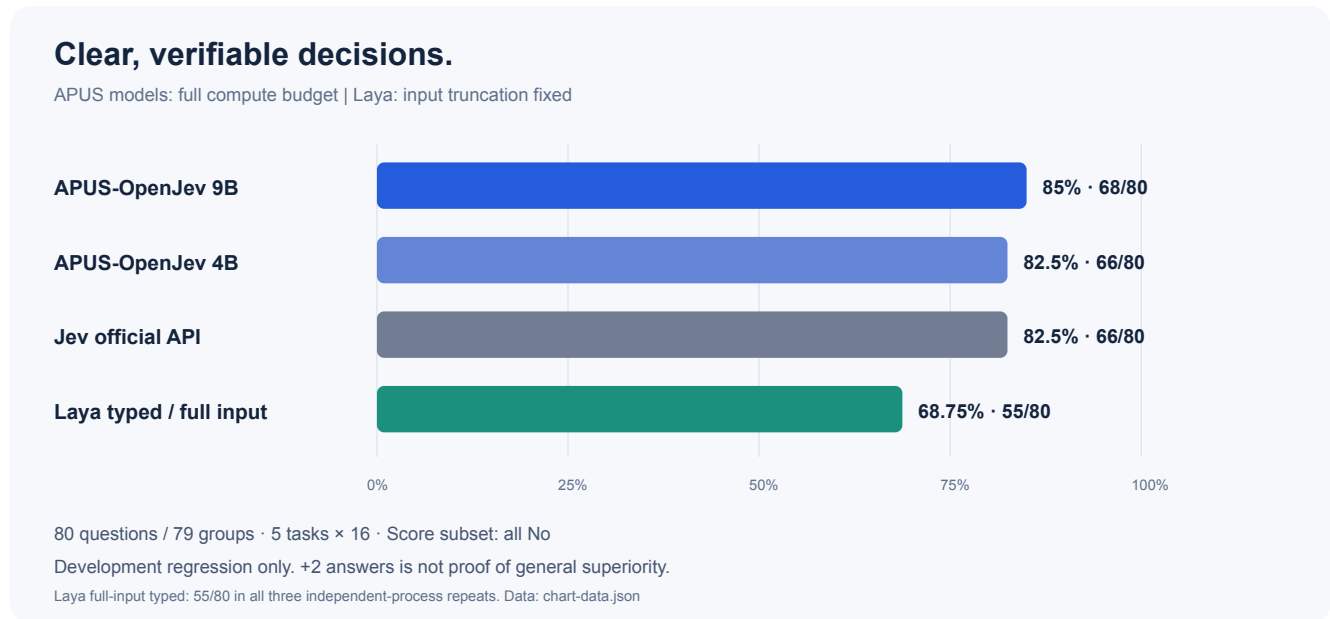


Figure 3. Decision accuracy on the same frozen panel. APUS results use the high compute budget; Laya uses the complete-input typed configuration after truncation was corrected.

System	Decision accuracy
APUS-OpenJev-v1 4B	82.50%
APUS-OpenJev-v1 9B	85.00%
Jev official API	82.50%
Laya	68.75%

These results use a frozen development comparison panel of 80 questions across 79 parent groups, with 16 questions each from Browser, HelpSteer3, BoolQ, MNLI, and Score. APUS results use the high compute budget; the 9B entry uses the selected intermediate version, and Laya uses its typed, complete-input evaluation with truncation corrected. The panel has been used during development and version selection. Its Score subset contains only No labels, and browser questions evaluate static candidate selection rather than completed browsing episodes. Historical latency results distinguish local inference from the official API's complete HTTP response.

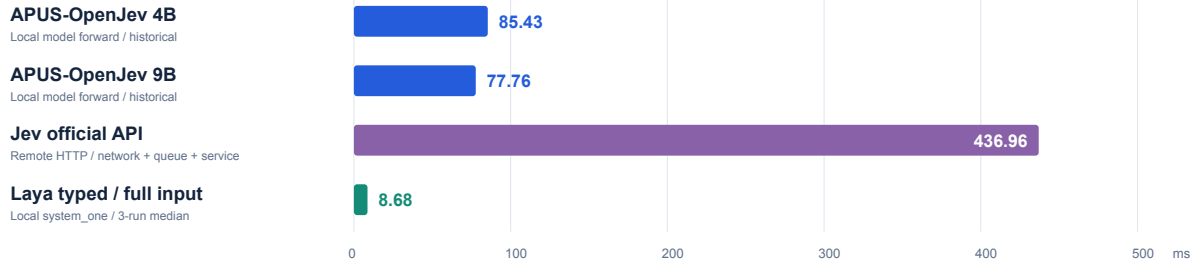
7.2 RESPONSE LATENCY

Response latency — see the numbers and their boundaries.

Same fixed 80-question panel · horizontal bars start at zero · lower measured time = shorter bar

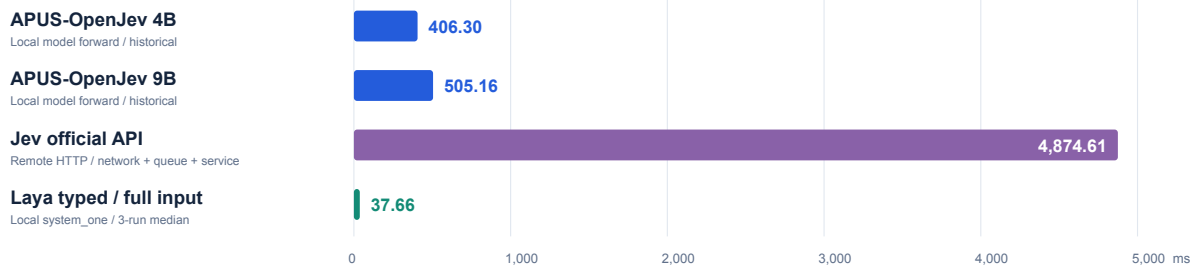
P50 · typical request

Linear scale: 0–500 ms



P95 · slower tail

Linear scale: 0–5,000 ms



Different timing scopes and runs: do not divide these bars to claim a speedup factor.

APUS: historical forward timings, not a new merged-package test. Laya: median of 3 per-run P50/P95 values.

P50 and P95 panels use different scales. Not text TTFT or tokens/s. Exact data, scope and source hashes: `chart-data.json`

Figure 4. Historical decision latency, showing P50 and P95 on separately labeled scales. Local model forward, local pipeline, and complete public HTTP response include different work; these observations do not establish a direct speedup ratio.

Measured path	P50	P95	Timing boundary
APUS-OpenJev 9B reference	77.76 ms	505.16 ms	Local model forward
APUS-OpenJev 4B reference	85.43 ms	406.30 ms	Local model forward
Laya typed, complete input	8.68 ms	37.66 ms	Local pipeline
Jev official API	436.96 ms	4874.61 ms	Complete public HTTP response

APUS timings come from the historical RTX PRO 6000 reference implementation. Laya includes tokenization, transfer, model execution, and output processing; its displayed statistics are the medians of three per-run P50 and P95 values. Jev includes network, queueing, and service overhead. The measurements describe complete decisions rather than streaming first-token latency or generated tokens per second. Exact values, source identities, and aggregation are retained in the accompanying `chart-data.json`.

7.3 EVALUATION AGENDA

The next evaluation phase should connect each architectural claim to an appropriate experiment. Depth learning requires matched initialization and data controls against full-depth-only training. Routing requires frozen policies, per-task risk–coverage curves, and a measured relationship between saved computation and errors. Shared-state execution requires numerical and decision-equivalence checks before latency tests. Browser capability requires replayable environments, action and TYPE evaluation, recovery behavior, and complete task outcomes. Jet-DCRL requires comparisons against continued supervision and exact utility optimization, with probability metrics reported alongside accuracy.

APUS-OpenJev-v1 establishes a concrete foundation for this program: language-conditioned decisions, shared parameters across execution budgets, and a training objective that connects those budgets. Its central direction is to make computational effort a controllable dimension of useful language-model behavior, evaluated through decisions and the workflows they enable.

Acknowledgments

We thank the Qwen team for the Qwen3.5-4B and Qwen3.5-9B base models, the ms-swift contributors for the training framework, and the creators of the open datasets used in this work.