Explicit versus learned search

Neural-guided tree search and search-trace learning

Angel Raychev
A technical retrospective on joint 2025 work with Yalcin Tur and Mihajlo Stojkovic.
Prepared 23 October 2026

Overview · Code

Overview. A language model can participate in search in two different ways: an external program can maintain a tree and ask learned models which branches to expand, or a model can learn to represent exploration inside its own generated sequence. I built a research pipeline connecting these approaches: batched neural-guided tree search, iterative policy and value training, and supervised learning from serialized search traversals. The common intermediate object is a search tree; different projections of that tree become different learning signals.

The experiments connect three levers: improving the learned models, allocating more inference-time search, and teaching a model to generate the search procedure itself. In the earlier Game of 24 study, separately selected search configurations reached reported accuracies of 29.39%, 53.44% and 54.58% for same-size 135M, 360M and 1.7B policy/value pairs. The later generalized-arithmetic study reported a budget-dependent crossover between externally controlled search and traversal-trained models. These observations motivate a joint view of learning and search rather than a single winner determined by model size.

The methods and experiments tell one connected story: an efficient external search engine supplies experience for policy learning, value learning and search-trace learning; the resulting models then change what can be achieved at inference time. The report follows that connection from implementation to empirical trade-offs.

Scope: a retrospective by Angel Raychev on joint 2025 work, preserving the original collaborators' credit. Numerical results and original plots are from the historical studies; no new model training or evaluation was performed. Different stages and search budgets remain explicitly distinguished.

1 The question is where search happens

Both approaches use neural networks. "Explicit" does not mean non-neural, and "learned" does not mean that the alternative has no learned components. The operational contrast is where branching decisions and revisitable state live.

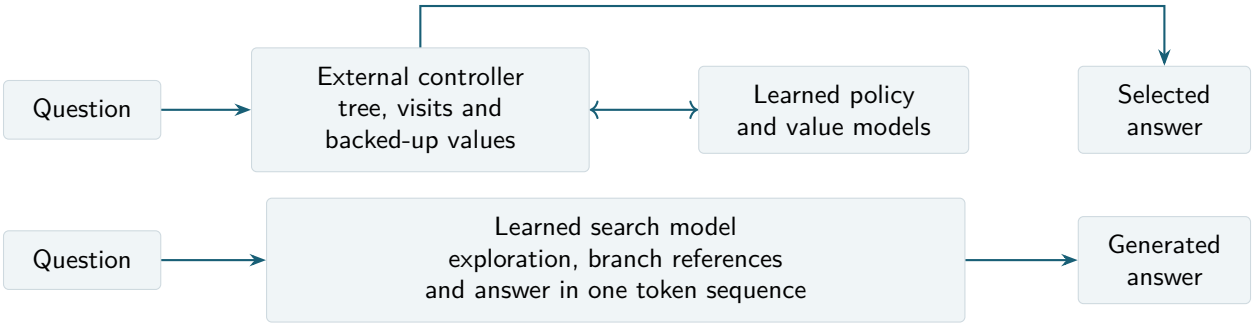


Figure 1. Two locations for search control and state. The upper answer is selected by the controller using model-guided statistics; the lower answer is emitted by the search-trace-trained model.

An external controller explicitly stores alternatives and statistics. A search-trace model represents exploration through text in a finite context. Learning can transfer work from repeated online search

into model parameters, but the training examples themselves may require substantial teacher-search computation. The useful comparison therefore concerns a joint allocation of model capacity, training effort and inference effort, not an isolated count of final-answer tokens.

2 Task representation and information boundaries

The project used arithmetic construction problems because the inputs, actions and final-answer conditions can be stated precisely. The earlier CS224N study used Game of 24; the later CS224R work generalized the target. A problem can be written

$$q = (X, t), \qquad X = \{x_1, x_2, x_3, x_4\}_{\text{multiset}}, \tag{1}$$

where a final expression must use each supplied number with its given multiplicity and evaluate to the target t using the permitted operations. The multiset matters: two occurrences of 3 are two resources, not one distinct value.

The language model proposes a textual step, rather than selecting from a hard-coded enumeration of all legal moves. For a question q and previous actions $a_1, \ldots, a_k$, the model-facing state is the textual prefix

$$s_k = (q, a_1, \ldots, a_k), \qquad a_{k+1} \sim \pi_\theta(\cdot \mid s_k). \tag{2}$$

An action might combine two remaining numbers and report the remainder. A separate final-answer marker identifies a terminal proposal.

```
Use 3, 3, 5, 12 to make 51.
3*12=36 Left:  3, 5, 36
3*5=15 Left:  15, 36
36+15=51 Left:  51
The answer is:  3*12+3*5=51.
```

Worked arithmetic illustration. This is not a new sampled rollout or a measured result. The same problem appears in the original search-trace example.

2.1 A learned value is not a correctness certificate

The value model estimates the promise of a prefix. It does not prove that the prefix's arithmetic or state bookkeeping is legal. In the historical implementation, partial states were model-generated text, and the terminal expression was checked separately. Describing the environment as precisely specifiable must not be confused with claiming that every intermediate step was enforced by a symbolic transition engine.

The distinction also separates two outcomes:

$$\text{discovery success}: \quad \exists \text{ a correct terminal candidate in the explored tree}, \tag{3}$$

$$\text{selection success}: \quad \text{the method's chosen final answer is correct}. \tag{4}$$

The first asks whether exploration found a solution. The second additionally tests whether the method recognized and selected it. Using an answer checker to choose a final candidate would change the information available to the method.

2.2 Training and evaluation have different roles for the checker

For data generation, terminal outcomes supply labels and identify successful trajectories for training. For the historical MCTS evaluation path, learned values guide search and terminal backups; the controller selects an answer before the final checker scores it. There is no basis for describing this evaluation path as an oracle that simply returns whichever explored answer is correct.

Key distinction. The environment supplies a task and a final correctness condition. The policy proposes, the value model estimates, the controller searches and selects, and the evaluator measures. These are distinct responsibilities even when they share data structures.

3 A custom policy–value search controller

The search engine stores a node's textual state, parent and children, visit count N, backed-up action value Q, and initial learned estimate v_ϕ. The policy and value model are separate services. The policy generates candidate continuations; the value model scores the resulting child prefixes. This separation permits independent model choices and training updates.

3.1 Selection, expansion and backup

The selection rule implemented in the inspected 2025 controller is

$$u^* = \operatorname*{arg\,max}_{u \in \text{children}(s)} \left[Q(u) + c \, \frac{\sqrt{N(s)}}{N(u)+1} \right]. \tag{5}$$

Here c controls exploration. The policy supplies sampled children, not a policy-prior multiplier in this formula. This is an AlphaZero-inspired system, not an exact reproduction of AlphaZero or canonical PUCT.

At an expansion, the policy supplies up to the configured branch factor b of candidate continuations. The adapter appends each candidate to its parent prefix and evaluates all resulting children with v_ϕ. New children start with zero visits and their learned estimate as the initial action value. A traversal backs a scalar z up the ancestry through running means:

$$N(s) \leftarrow N(s) + 1, \qquad Q(s) \leftarrow Q(s) + \frac{z - Q(s)}{N(s)}. \tag{6}$$

The source of z matters: learned estimates evaluate nonterminal states; terminal outcomes supply data-generation labels. The evaluation subclass instead backs up learned terminal estimates. Shared tree machinery thus supports different collection and measurement semantics without an answer-selection oracle.

The implementation searches up to configured expansion budgets and can return results at multiple budget checkpoints while continuing the same tree. At the end, it follows maximum-Q children to a leaf. That actual final-choice rule is distinct from the exploration rule in Equation (5) and from a most-visited-child rule.

3.2 The adapter is part of the algorithmic contract

A model interface returns candidate states paired with values. It batches policy requests, flattens children for value inference and maps replies back to each parent. Model serving and tree statistics remain separate responsibilities.

The interface separates proposal quality and diversity from value quality; policy capacity from evaluator capacity; branching breadth from expansion count; and local model scores from the controller's accumulated evidence. More children also create more evaluation work. A value model can misrank a correct continuation. Extra search effort is not useful progress by definition; the complete system must be measured.

Implementation anchors: `utils/mcts_base.py`, `utils/policy_value.py`, and `evaluate/mcts_evaluator.py`. These links point to the historical source snapshot, not to an unrecorded moving branch.

4 Batching a forest instead of waiting on one tree

Individual searches have irregular control flow: one prefix needs expansion, another is waiting for a response, and a third has reached a terminal answer. Running each question as a completely isolated sequence of tiny model requests would couple inference efficiency to that irregularity. The forest abstraction collects expansion work from many trees into a common queue.

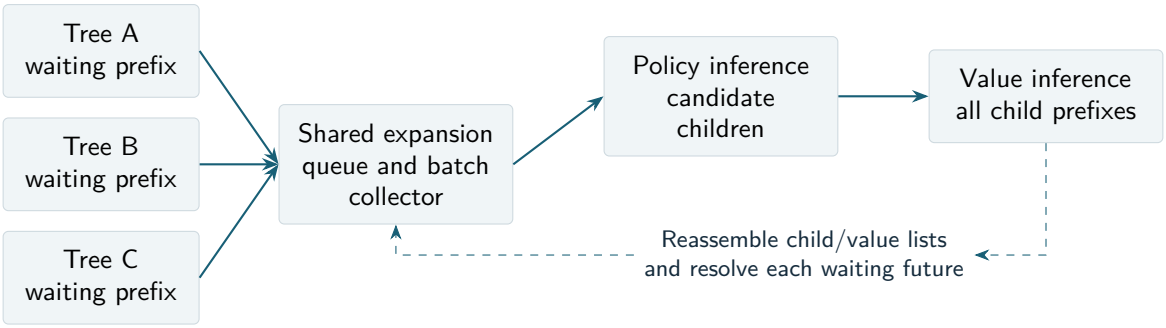


Figure 2. The forest converts independent search-control flows into batched policy and value work, then returns each result to the tree that requested it.

Each tree enqueues a question/prefix pair with a future. The collector forms batches, calls the policy–value adapter and routes each returned list to the matching future. Tree slots then continue traversal or pick up another question. The source allocates tree slots separately from model batch size (specifically, twice that batch size), so the forest can hold more waiting searches than fit in one inference batch.

There are two distinct batching operations. First, several parent prefixes are presented to the policy together. Second, all sampled children from those prefixes are flattened into a value batch. The latter batch may be substantially larger than the former. Reassembly preserves the correspondence between each parent, its proposed children and their estimates.

4.1 Scheduling, batching and hardware utilization

The forest uses asynchronous tasks and futures to express independent tree progress, while the historical service adapter uses synchronous HTTP calls. Batching is explicit; hardware throughput and network overlap are separate execution properties to measure.

The pipeline has an explicit unit of scheduling: an expansion request, rather than an entire problem or an individual output token. It also separates three things often collapsed into "batch size": the number of live questions, the number of parent prefixes in a model call, and the number of child prefixes to be evaluated.

The broader reusable pattern is a queue between irregular control flow and regular batched inference. The same idea is relevant whenever an agent's next action depends on previous results, while the expensive neural operations benefit from grouping work across independent tasks.

5 One tree produces three training datasets

The search tree is not only an inference artifact. The generator projects it into three different supervised products. Keeping those products explicit is one of the most useful parts of the implementation: they preserve different information and should not be described as interchangeable "synthetic data."

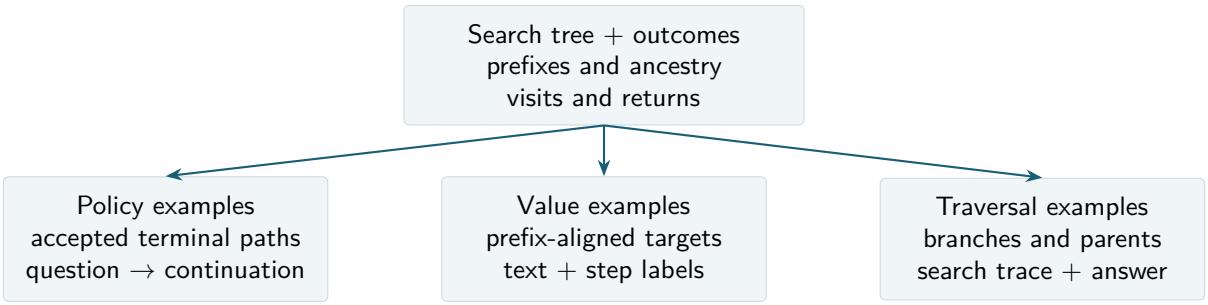


Figure 3. Three projections of the same search process. A successful path teaches a continuation; a return-labeled prefix teaches an estimate; a traversal teaches how exploration is represented.

Product	Stored contract	Information retained
Policy	Question and full accepted terminal continuation	A successful path, including its textual steps and final expression.
Value	Question plus trajectory text, with an ordered label list	Targets aligned to the question/step boundaries along each terminal path.
Traversal	Question, first-visit trace and selected accepted answer	Alternative branches, explicit node/parent identifiers and a final answer.

5.1 Policy learning from accepted paths

Let $\mathcal{D}_+$ contain terminal trajectories accepted by the collection-time checker. The policy objective is ordinary supervised next-token learning on those trajectories:

$$\mathcal{L}_{\text{policy}}(\theta) = -\mathbb{E}_{(q,\tau)\sim\mathcal{D}_+} \log \pi_\theta(\tau \mid q). \tag{7}$$

The selection mechanism matters: these are paths discovered by the current search process, not samples from an independently specified optimal-policy distribution. Updating the models changes which paths the next collection round is likely to produce. The implementation thus connects search-assisted data collection to iterative supervised improvement.

Before iterative search-assisted training, synthetic worked examples bootstrap the policy's step format and final-answer convention. That initial supervised stage gives the search controller a usable proposal interface; subsequent rounds replace fixed formatting examples with successful paths discovered by the evolving policy–value system.

5.2 Value targets summarize observed returns

For a node s, let $\mathcal{R}(s)$ be the multiset of terminal returns actually backed up through it during data collection. When that multiset is nonempty, the generator uses

$$\widehat{v}(s) = \frac{1}{|\mathcal{R}(s)|} \sum_{r\in\mathcal{R}(s)} r. \tag{8}$$

It falls back to the current terminal path's label when no backed-up labels are present. Consequently, intermediate labels can be fractional even though each terminal outcome is binary. They are search- and visitation-dependent empirical targets, not true optimal values or exhaustive averages over every reachable solution.

5.3 Step-value learning without replacing the language-model head

The later implementation makes the label/output alignment concrete. It retains a causal language model's vocabulary head, takes the logit corresponding to the token `1`, and trains that scalar with a binary-cross-entropy objective. If $\ell_\phi(s_j)$ is that designated logit at a labeled boundary, then

$$v_\phi(s_j) = \sigma(\ell_\phi(s_j)), \qquad \mathcal{L}_{\text{value}} = \frac{1}{\sum_j m_j} \sum_j m_j \ \text{BCEWithLogits}\left(\ell_\phi(s_j), y_j\right), \tag{9}$$

where m_j selects supervised boundary positions and y_j is the corresponding return target. This is a sigmoid of one vocabulary logit, not the softmax probability that the model would generate the token `1`.

The tokenizer identifies tokens whose decoded text contains a newline. The dataset transformation aligns the ordered label list to those positions. Nonlabel tokens and padding receive a sentinel value and are masked out of the loss. The model wrapper checks that the newline mask and label mask agree, rather than silently training against shifted or missing labels.

At inference, the value service takes the same designated vocabulary logit at the end of a supplied prefix and applies a sigmoid. Keeping the value readout inside the ordinary language-model head makes the later value model compatible with the vLLM-based serving path. The important design is the agreement among data boundaries, loss mask, readout position and inference-time extraction.

5.4 The value interface evolved with the system

The earlier SmolLM2/Game of 24 implementation uses a separate nonlinear scalar head served with PyTorch/FastAPI. The later Qwen-based source instead uses the vocabulary-logit readout described above, supporting vLLM-based serving. The learning/search interface carries across the two stages, while the readout implementation and task distribution change. Table 1 records the experimental setups together.

Implementation anchors: `generator`, `step-value trainer`, `value server` and `policy trainer`.

6 Learning a search traversal, not only its successful path

A successful trajectory omits the alternatives that were considered along the way. Stream-of-Search-style supervision retains more of that process: the generator records nodes in first-visit order and serializes references to their parents or ancestry. A model can then be trained on text that contains branch switching, unsuccessful avenues and a selected final answer.

For the running arithmetic example, successful and unsuccessful branches can coexist:

```
Q | Use 3, 3, 5, 12 to make 51.
<START_THOUGHT>
N1->Q | 3*12=36 Left:  3, 5, 36
N2->Q | 12*5=60 Left:  3, 3, 60
N3->N1 | 3*5=15 Left:  15, 36
N4->N2 | 3*3=9 Left:  9, 60
N5->N3 | 36+15=51 Left:  51
N6->N4 | 60-9=51 Left:  51
N7->N2 | 60-3=57 Left:  3, 57
N8->N7 | 57-3=54 Left:  54
<END_THOUGHT>
<START_ANSWER>
3*12+3*5=51
<END_ANSWER>
```

Simplified illustrative serialization, not a new model output or an exact byte-for-byte saved traversal. The generator's terminal entries may include the full ancestor chain. Two branches reach 51; the branch through `N7` and `N8` reaches 54 instead, preserving an unsuccessful avenue alongside the selected answer.

The parent identifiers make the distinction from a linear solution explicit. Node `N4` continues `N2`, not the immediately preceding textual line `N3`. The next generated token can therefore describe an action in a different branch while the sequence as a whole remains linear.

6.1 What the supervision asks the model to absorb

The traversal model is fine-tuned on teacher-search traces and answers. It starts from a pretrained language model; "a fresh model" here does not mean training randomly initialized weights. The target is a representation of the teacher's exploration process, including its ordering and branch references, rather than a guarantee that every teacher decision is optimal.

Search-trace learning can amortize the cost of exploration across future questions. It may also inherit the teacher's limitations and the trace distribution's conventions. More trace tokens are not automatically more useful supervision: branch diversity, the recoverability of state, sequence length and the quality of the appended answer all influence what can be learned.

6.2 Rejection self-training after traversal supervision

The project also implements a STaR-style self-training path: generate candidate completions on new questions, retain accepted completions and use them in a subsequent training dataset. The scripts can revisit unresolved questions over repeated generation rounds. That is a concrete generate–verify–train loop; it should not be equated with every detail of the original STaR algorithm, including its answer-conditioned rationalization step [6].

The external controller and the learned traversal model are therefore connected by more than a shared benchmark. Search builds a teacher distribution; serialization determines what the student sees; supervised learning and rejection self-training determine how that experience is absorbed into later generation.

7 Experiments and empirical results

The experiments progressed through two stages. The CS224N study examined how policy capacity, value capacity, iterative training and inference search interact on Game of 24. The CS224R extension generalized the target and asked how the externally managed search compares with models trained on its traversals. The stages share a learning/search architecture, but not one task distribution or checkpoint family.

7.1 Experimental recipes

	Game of 24 study	Generalized-arithmetic study
Model family	SmolLM2, independently varied 135M, 360M and 1.7B policy/value models	Reported Qwen2.5 1.5B policy and 1.5B value; 3B traversal model
Task	Combine four supplied numbers to reach 24	Four integers in $[1, 12]$, target in $[1, 999]$
Collection search	Branch factor 3, up to 20 expansions, exploration constant 0.3	Branch factor 5, up to 25 expansions, exploration constant 0.3
Learning	Format SFT, followed by successful-path policy training and positive/negative value training	Iterative policy/value training, then traversal SFT and rejection self-training
Evaluation focus	Capacity matrix, training rounds, breadth/expansion sweeps	MCTS checkpoints and SoS/STaR comparisons across traversal/search settings

Table 1. The recipes described in the original reports. Collection settings are not the larger inference-search sweeps. Model sizes in the final CS224R account differ from earlier 0.5B development runs retained in the repository.

The later report describes a pool of 105,000 problems with a 99,000/3,000/3,000 train/validation/test split. Policy formatting is bootstrapped with synthetic worked solutions. Subsequent rounds collect new terminal trajectories and train the policy on accepted paths and the value model on labeled prefixes. The collection recipe samples five candidate continuations at temperature 1 and filters duplicates.

For traversal learning, the final policy/value teacher is frozen. The report describes 12,000 fresh questions, traversal collections with branch factors $\{1, 2, 3, 5, 8, 13, 21\}$, and a separately fine-tuned search model for each setting. Traversal SFT uses 16 epochs, learning rate 2×10^{-5} and batch size 64. The two reported rejection self-training variants use one attempt per question (STaR) or up to ten attempts, retaining the first accepted completion (STaR*). These are the historical recipes, not newly executed runs.

7.2 Proposal and evaluation capacity interact

The earlier study independently varied both components, producing a full nine-configuration matrix rather than only scaling one monolithic model. Table 2 preserves the numerical entries from the original LaTeX.

	Policy size		
Value size	135M	360M	1.7B
135M	12.29 ± 1.40	20.31 ± 0.56	18.93 ± 1.40
360M	15.95 ± 1.30	26.35 ± 0.99	25.42 ± 1.04
1.7B	10.53 ± 1.47	24.20 ± 2.30	26.87 ± 1.55

Table 2. Reported Game of 24 accuracy (%) at the study's common search setting. Rows vary value capacity; columns vary policy capacity. The original mean ± SD notation is retained without reinterpreting the dispersion as independent random seeds or confidence intervals. Source: CS224N report, capacity table [1].

For the 135M policy, moving from the 135M to the 360M value model raises reported accuracy from 12.29% to 15.95%. Increasing the value model further to 1.7B instead gives 10.53%. With the 1.7B policy, the largest value model yields the highest entry in that column, 26.87%. Increasing evaluator capacity is therefore not uniformly beneficial across proposal models in these observations.

The matrix exposes a systems-level design choice. The policy determines which candidates exist to be evaluated; the value model influences which of those candidates receive further exploration. Increasing one component's capacity does not by itself determine the quality of that interaction.

7.3 Training improves the search curve

Figure 4 shows that gains were not obtained only by growing a fixed model's tree. Iteration 2 of the 135M/135M system lies above iteration 1 over the displayed range: updating from search-generated trajectories improves the system that performs the next search.

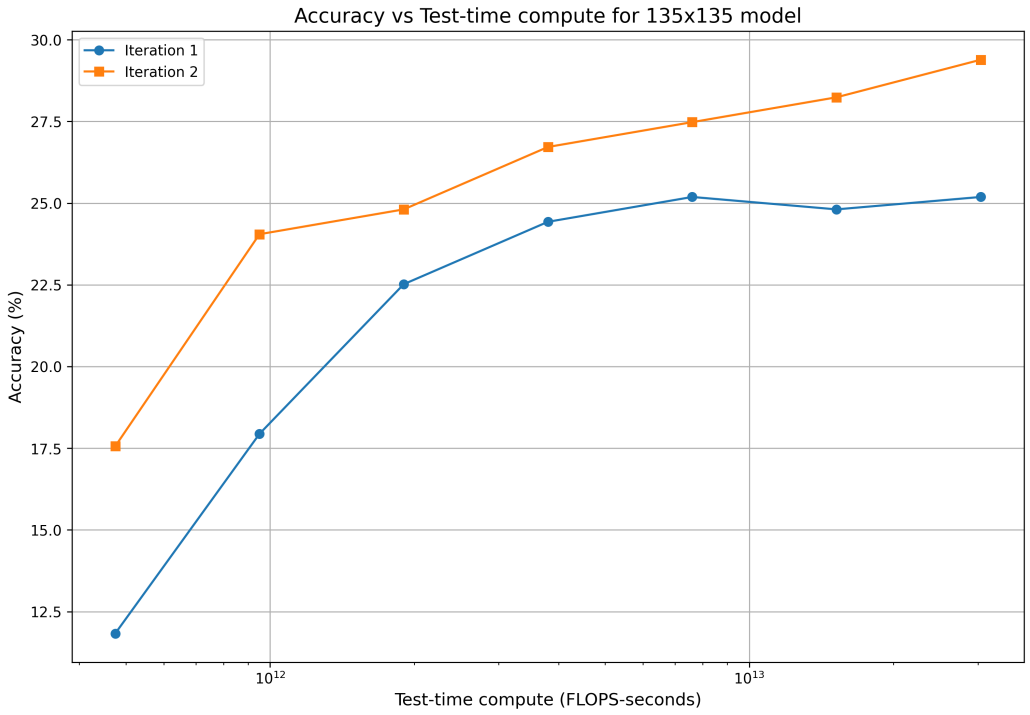


Figure 4. The original Game of 24 training-round figure, recovered from Overleaf. Both policy and value are 135M. The plotted comparison is iteration 1 versus iteration 2; it is not an iteration-0 curve. The x-axis retains the original estimated-compute convention and its historical label.

Figure 5 gives the corresponding size/search view. The reported curves show the 360M and 1.7B systems benefiting over a broader explored range than the first-round 135M system. Table 3 records the separately selected peak configurations, including the later 135M result; the 29.39% peak should not be confused with the lower first-round 135M curve in Figure 5.

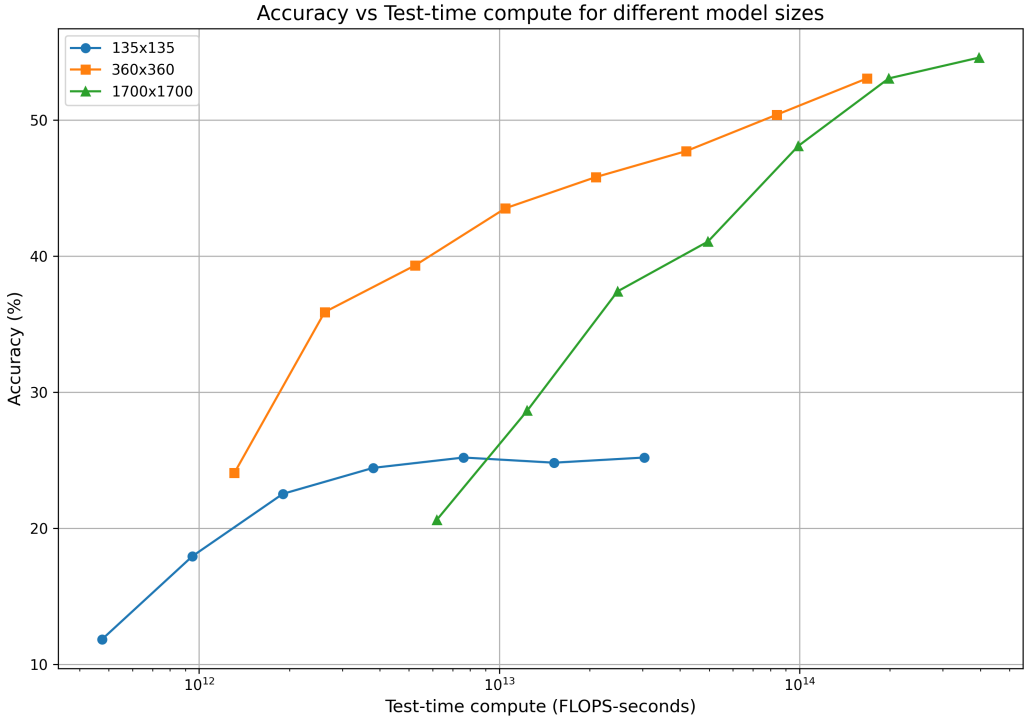


Figure 5. Original same-size policy/value search sweeps from the CS224N study. Each legend entry names both model sizes in millions of parameters. The horizontal coordinate is the report's estimated resource measure, not a new measurement of end-to-end inference cost.

Policy / value	Accuracy (%)	Branch factor	Expansions
135M / 135M	29.39	16	64
360M / 360M	53.44	20	102
1.7B / 1.7B	54.58	13	79

Table 3. Highest reported accuracy and its configuration in the CS224N appendix. All three use temperature 1.0 and exploration constant 0.3. These are separately chosen search settings, not matched-compute points or a claim that model capacity is irrelevant.

Training and inference search are thus complementary levers. A better learned policy/value pair can shift the curve; a larger search budget explores more alternatives using that pair. The broad empirical question is how to allocate effort across both, rather than whether either learning or search can be removed.

7.4 External search versus internalized traversal

The later study applies the same learning/search cycle to generalized targets. Figure 6 compares successive MCTS checkpoints. The final report records 51.13% accuracy at its final checkpoint over the wider explored search range.

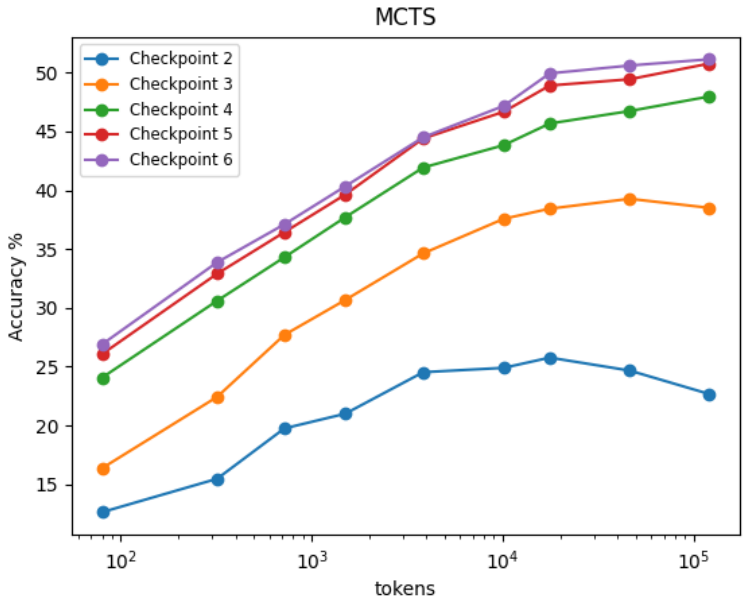


Figure 6. Original CS224R MCTS checkpoint curves, recovered from the final-report Overleaf project. The axes retain the original token-budget convention. This is a separate experiment stage from the SmolLM2/Game of 24 figures above.

The central comparison is Figure 7. A traversal-trained model can use a single generated sequence to represent alternatives that the external system maintains in memory. The reported SoS and self-trained variants exceed MCTS over much of the lower and middle plotted range. At the largest displayed setting, external MCTS overtakes them while the contextual variants decline. The displayed crossover is the finding; it does not require declaring a universal winner.

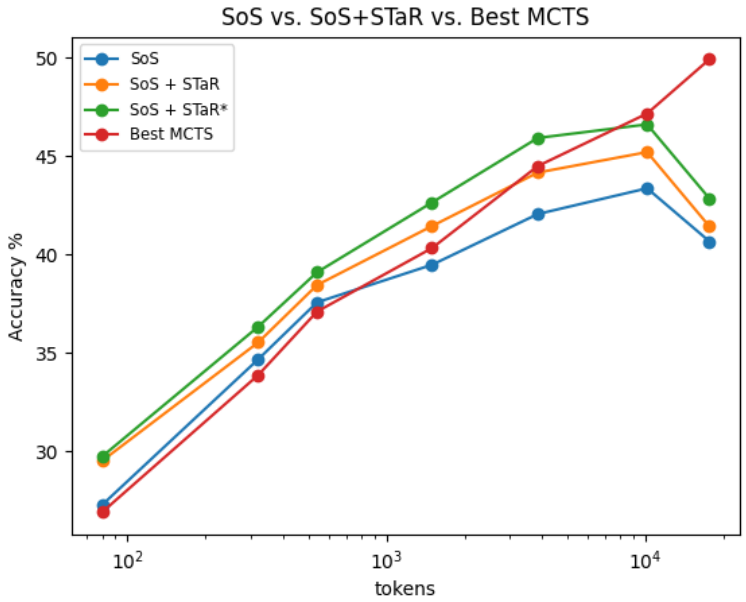


Figure 7. The original final-report comparison of MCTS, SoS and two rejection-self-training variants. Different SoS points correspond to models trained with different traversal settings, not necessarily to stopping one frozen model at different budgets. The rightmost MCTS point here is not the separate wider-range 51.13% result. Source: original CS224R Figure 2 [2].

The original discussion proposes context length as an explanation for the long-trace decline. The plot establishes the observed decline, but not its cause. Longer traversals also change the supervision and generation workload. Distinguishing those mechanisms would require a separate controlled experiment.

7.5 What the combined evidence tells us

Three observations carry the main message. First, proposal and evaluation are independently trainable components whose capacities interact. Second, iterating on search-generated experience can improve the next search, not just its final answer format. Third, teaching the traversal creates a different resource/performance profile from maintaining the tree externally. These are concrete empirical and engineering results even though the larger efficiency/generalization question remains open.

Interpretation of the historical figures. These are the reported 2025 results, recovered from the original LaTeX and image assets. Refreshed arithmetic checks do not constitute a rerun or retrospective rescoring. The historical axes include resource proxies rather than newly measured end-to-end FLOPs; equal total parameter counts do not equalize neural calls, prefix processing or teacher-search cost. The observations do not establish a universal or asymptotic separation. Appendix A records the remaining provenance distinctions.

8 Discussion: a joint allocation of learning and search

The useful resource question has two timescales. During construction, search produces successful paths, value targets and traversal supervision. During use, a trained system either calls learned models inside an external controller or emits its own search representation. A useful accounting separates

$$C_{\text{total}}(n) = C_{\text{training}} + C_{\text{teacher search}} + \sum_{i=1}^{n} C_{\text{inference}}(q_i). \tag{10}$$

This is an accounting frame, not a fitted cost model. The teacher-search investment can be amortized across later questions, but it must still be counted when comparing complete learning systems.

The external controller offers explicit branch identity, revisitable state and replaceable selection rules. The internalized model conditions each new decision on its preceding search sequence. Those interfaces have different strengths, and hybrids can retain explicit state while learning more of the control. A fair continuation would specify the resource or generalization contrast first, then hold the remaining information and training conditions fixed.

For a finite arithmetic task with complete legal enumeration, exhaustive search can in principle find an existing solution. The scientific issue is therefore not escaping finite enumeration. It is how effectively the system spends limited computation, what its training amortizes, and how well that behavior transfers beyond the training distribution.

9 Artifacts, reuse and contribution

The companion repository contains the search engine, policy/value interface, data generator, trainers and experiment scripts. Historical source is linked at the June 2025 snapshot throughout this report. Public policy, value and search-trace model artifacts make the training pipeline inspectable beyond its implementation: readers can inspect experimental weights, configurations and model cards.

The companion materials now bring the methods, original result figures and numerical tables into one current report. The repository also retains an independently identified archive of earlier evaluation records and plotting data, rather than silently merging them with the final-report figures. Local CPU integrity checks exercise defined data/verification boundaries; model-backed execution additionally requires the appropriate data, model access, serving environment and GPU resources.

The source package builds this report with its original empirical figures and vector architecture diagrams, without model inference. The evidence archive identifies historical records by source commit and blob, preserving actual model IDs and duplicate-snapshot relationships. Its inspection and plot-recreation tools operate only on retained data. Reproducing the full training study additionally requires the original splits, compatible model/tokenizer revisions, serving environment and resource configuration.

9.1 My contribution and the original collaboration

The work began as a joint Stanford CS224N project by Yalcin Tur, Mihajlo Stojkovic and me. The original report credits me with MCTS implementation, evaluation and writing; it credits Yalcin and Mihajlo with policy/value training, data generation and infrastructure for that stage.

I subsequently extended the work for Stanford CS224R. That report identifies me as lead developer, credits most of the later codebase, repeated MCTS refactors and the experiments, and retains Yalcin and Mihajlo's credit for the initial code. My later engineering emphasis was the end-to-end pipeline: efficient batching across searches, iterative model/data flow, step-value training and the search-trace branch. Anjiang Wei and Rafael Rafailov are named as mentors in the CS224N report.

This retrospective is authored by me. The two original reports retain their original authorship and identity.

Key distinction. The reusable idea is to treat search as both an inference procedure and a producer of structured learning signals. Which parts should remain in an external controller, which should be represented in context, and which should be amortized into model parameters is a joint systems-and-learning question.

Resources. Project article and report; companion repository; and public model artifacts.

A Additional empirical detail and source record

A.1 Breadth and repeated expansion

The original branch/expansion sweeps vary two different decisions: how many candidates an expansion proposes, and how many expansion steps the controller takes. More expansions are not the same as greater arithmetic solution depth. Figure 8 retains the original contour views; the red points mark measured configurations and the colored surfaces interpolate between them.

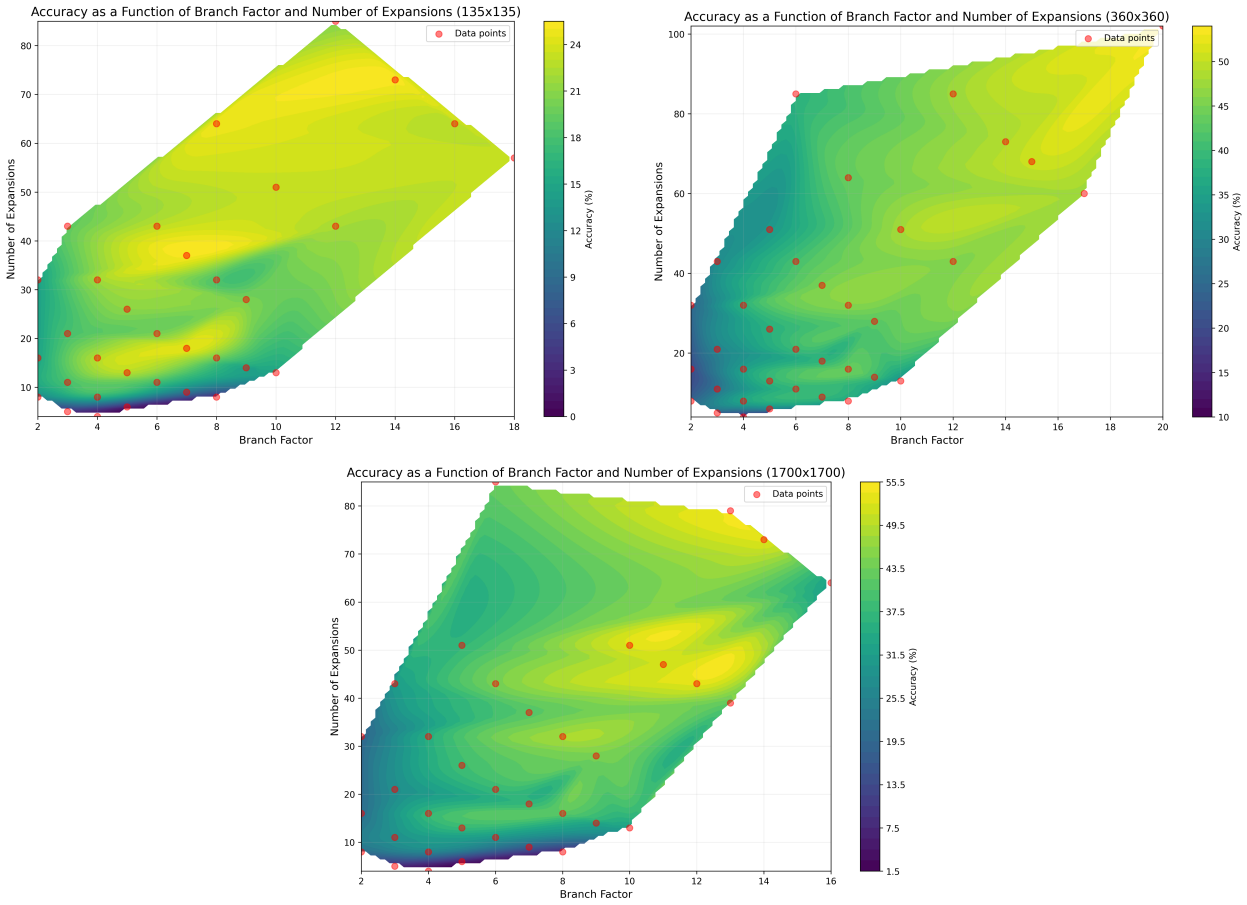


Figure 8. Original Game of 24 branch/expansion plots for the 135M, 360M and 1.7B same-size policy/value systems, respectively. Interpolated regions are visual summaries, not additional measured runs. Recovered from the CS224N Overleaf project.

Branches	Expansions	135M / 135M	360M / 360M	1.7B / 1.7B
2	8	11.83	24.05	20.61
3	21	17.94	29.77	27.86
4	16	22.52	34.73	37.02

Table 4. Selected same-search-setting accuracies (%) transcribed from the original CS224N appendix. Matching branch/expansion counts does not match neural compute across different model sizes. Temperature is 1.0 and exploration constant 0.3.

A.2 How the consolidated evidence was assembled

The original Overleaf projects supplied the numerical tables and unmodified plot images. The original reports retain their author order and identity; this consolidated retrospective credits that work while explaining the implementation and results together.

The original CS224N split description contains an unresolved counting inconsistency, so this report does not invent a corrected evaluation denominator. Its capacity table's SD is retained as originally reported; the surviving source does not establish a new random-seed interpretation. The later report's declared model sizes and recipes are attributed to that report, rather than inferred from similarly named public checkpoints.

The older Git plotting script and timestamped evaluation records are useful additional evidence, but belong to earlier configurations. In particular, the older plotting arrays have different peaks and fewer contextual-search points than the final CS224R figures. They are archived separately with commit provenance. No curve has been numerically digitized from an image, and no final-report datapoint has been manufactured by joining those different records.

Research context and primary references

The work sits at the intersection of neural-guided external search and learned representations of search. TS-LLM studies AlphaZero-like tree search for language-model decoding and training [4]. Stream of Search provides a direct precedent for encoding exploration as a flattened language sequence [5]. STaR supplies the broader generate–verify–train self-improvement context [6]. Schultz and colleagues explicitly contrast external and internal planning with language models in board games [7]. These connections explain the project's framing; this retrospective does not claim that either broad approach was introduced here.

References

[1] Yalcin Tur, Mihajlo Stojkovic and Angel Raychev. *Size Doesn't Matter If You Search Enough*. Stanford CS224N custom project report, 2025.

[2] Angel Raychev, Yalcin Tur and Mihajlo Stojkovic. *In-context Search: Efficiency Boost or Fundamentally New Capability?* Stanford CS224R final report, 2025. Public Stanford copy.

[3] Angel Raychev, Yalcin Tur and Mihajlo Stojkovic. *Classical versus contextual search*. Project source, historical snapshot `1a7fda1`, June 2025; refreshed companion materials available from the companion repository.

[4] Xidong Feng and colleagues. *AlphaZero-Like Tree-Search Can Guide Large Language Model Decoding and Training*. 2023. arXiv:2309.17179.

[5] Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma and Noah D. Goodman. *Stream of Search (SoS): Learning to Search in Language*. 2024. arXiv:2404.03683.

[6] Eric Zelikman, Yuhuai Wu, Jesse Mu and Noah D. Goodman. *STaR: Bootstrapping Reasoning With Reasoning*. 2022. arXiv:2203.14465.

[7] John Schultz and colleagues. *Mastering Board Games by External and Internal Planning with Language Models*. 2024; revised 2025. arXiv:2412.12119.