State-Driven Go Opening Commentary: Harnessing State Machines and Retrieval-Augmented Multi-Agent LLMs

Shi-Jim Yen[1], Yung-Jen Lo[1]

Abstract— *Although recent advancements in Go Artificial Intelligence (Go AI) have achieved milestone victories over human world champions by focusing on playing strength search and reinforcement learning, Go Mentor systems capable of providing natural language game commentary remain an open gap. Existing attempts largely rely on general-purpose Large Language Models (LLMs). However, naive auto-regressive LLMs face severe spatial perception bottlenecks and potential hallucinations when parsing two-dimensional (2D) Go board states. To address these limitations without computationally expensive retraining, this study proposes a state-driven multi-agent Go opening commentary framework that integrates rigid Harness state machine control with a Trie-Structured Hierarchical Retrieval-Augmented Generation (Trie-S RAG) framework. Powered by Qwen3.5-27B as the backbone LLM, we implement structured exact retrieval and recursive backtracking of Kogo's Joseki Dictionary using a Trie-structured hierarchical tree matching Go game tree topology. Furthermore, we incorporate an NLI-based hypothesis verification module ($P \models H$) to guarantee that generated claims are strictly entailed by retrieved authoritative reference text. Comprehensive experiments across a 7-stage progressive ablation study evaluated on 10 classic opening layouts by dual frontier LLM judges (Claude 3.5 Sonnet and Gemini 1.5 Pro) demonstrate that our full proposed system achieves a superior overall score of 4.50 out of 5.00 (outperforming the Naive SGF baseline score of 2.02). Additionally, token efficiency benchmarks confirm a 51.8% reduction in inference latency (a 2.07× speedup) under comparable token budgets.*

Keywords— Go Opening Commentary, Large Language Models, Retrieval-Augmented Generation (RAG), Trie-S RAG, Harness State Machine, Multi-Agent Systems, Natural Language Inference (NLI), Ablation Study.

1 Introduction

In recent years, Game AI powered by deep reinforcement learning and Monte Carlo Tree Search (MCTS), such as AlphaGo [1] and KataGo [2], has surpassed top human grandmasters. However, these systems operate primarily as numerical "black boxes" outputting win-rate probabilities and principal variation arrays. While effective for move generation, they lack the capacity to articulate human-comprehensible tactical explanations and pedagogical insights for human learners.

Conversely, applying general-purpose Large Language Models (LLMs) to Go commentary generation has emerged as a promising direction. However, naive auto-regressive LLMs face severe spatial perception bottlenecks when parsing two-dimensional (2D) Go board states from sequential one-dimensional (1D) move strings. This spatial blindness leads to severe coordinate hallucinations, step-number confusion, and ungrounded tactical claims. Furthermore, standard dense vector Retrieval-Augmented Generation (RAG) models [3] often suffer from off-board chunk pollution, retrieving unrelated joseki variations that degrade commentary quality.

To overcome these fundamental challenges without resorting to computationally expensive model fine-tuning, this paper presents a state-driven multi-agent Go opening commentary framework. The primary contributions of this work are summarized as follows:

1. **Go Board State Analyzer**: A deterministic rule engine incorporating Sabaki pattern matching with 8-fold geometric symmetry and KataGo GTP engine integration to resolve spatial perception bottlenecks.
2. **Trie-Structured Hierarchical RAG (Trie-S RAG)**: A tree-structured index matching Go game tree topology for exact, off-board-pollution-free retrieval of Kogo's Joseki Dictionary (KJD) [4].
3. **NLI Hypothesis Verification Module**: An entailment verification mechanism using `mDeBERTa-v3` ($P \models H$) to audit tactical claims before passing them to downstream generation agents.
4. **Harness State Machine & Multi-Agent Orchestration**: Rigid externalized working memory control [5] coupled with specialized agent role separation (Go Sensei, Lecturer, and Translator Agents) using `Qwen3.5-27B` as the backbone model.
5. **7-Stage Progressive Ablation Study & Benchmark**: Systematic quantitative evaluation across 7 progressive ablation steps using dual frontier LLM judges, proving the individual efficacy of every architectural component.

2 Go Feature Extractor & Spatial Perception

To resolve the spatial understanding bottleneck inherent in LLMs, we designed a Go Board State Analyzer comprising three core components:

2.1 PureGoBoardEngine & Sabaki Pattern Engine

The `PureGoBoardEngine` is a lightweight Python rule engine that maintains the 2D 19×19 board state in memory, calculating liberties and tracking captures. Coupled with `GoTacticAnalyzer`, it applies Sabaki

[1] Department of Computer Science and Information Engineering, National Dong Hwa University, Hualien, Taiwan

© 2026 Information Processing Society of Japan

pattern-matching rules across 8-fold geometric symmetry transformations (C_{4v} point group operations, including 4 rotations and 4 reflections) to identify local tactical shapes (e.g., Atari, Hane, Cut, Extension, Tengen, Hoshi) regardless of corner orientation.

2.2 Golden Standard Board Trajectory (`joseki_sequence_human`)

The analyzer translates raw coordinate SGF strings into `joseki_sequence_human`, an absolute physical truth anchor formatted as:

```
[Step]. [Color][[Coord (Quadrant)]]_[Tactic Tag]
```

For example: `3. B[P3 (Bottom-Right)]_Komoku Small-Knight Enclosure`. This immutable representation eliminates coordinate hallucination at the input level.

2.3 KataGo GTP Engine Integration

The analyzer connects directly to KataGo via the GTP protocol, computing objective numerical features:

- **Win-rate (Winrate)**: Current player's winning probability (e.g., 50.95%).
- **Score Lead (Lead)**: Estimated point lead/deficit (e.g., −0.23 pts).
- **Win-rate Delta (ΔWinrate)**: Positional impact of the executed move.

3 Trie-S RAG & NLI Verification

3.1 Trie-Structured Hierarchical RAG (Trie-S RAG)

Standard dense vector RAG frequently retrieves off-board joseki chunks due to semantic similarity across similar Go terminology (e.g., Star point joseki chunks mixed into Komoku queries). To eliminate this retrieval pollution, we structure Kogo's Joseki Dictionary (KJD) as a hierarchical Trie matching the game tree topology $\mathcal{T} = (V, E)$. Each node $v \in V$ represents a board state, indexed with `moves_string` and `parents_ids`. When a query is initiated, Trie-S performs exact tree matching and Depth-First Search (DFS) path backtracking, retrieving only the exact local variation sub-tree.

3.2 NLI Hypothesis Verification Module

To guarantee that generated commentary is factually grounded in authoritative reference literature, we integrate a Natural Language Inference (NLI) verification module powered by `mDeBERTa-v3-base-mnli-xnli`:

- **Premise (P)**: The linear text extracted from retrieved KJD documents along the matched path (`linear_story_premise`).
- **Hypothesis (H)**: The tactical claim/opinion synthesized by the Sensei Agent (`claim`).

The model computes $P \models H$. If the output label is `entailment`, the claim is verified as `VERIFIED_TRUE`. If `neutral` or `contradiction`, a warning/failure status is recorded in the Harness log, instructing the downstream Lecturer Agent to prioritize factual KataGo evaluations over ungrounded claims.

4 Harness State Machine & Multi-Agent Orchestration

4.1 Harness State Machine Control

Inspired by Harness-1 [5], working memory, curated RAG chunks (C_t), document caches (D_t), and verification logs (V_t) are externalized into an explicit state transition matrix $S_t = (H_t, C_t, D_t, V_t)$. The state machine enforces a rigid 3-turn operational workflow:

- **Turn 1**: Mandatory execution of `hierarchical_trie_retrieval` to populate memory.
- **Turn 2**: Synthesis of tactical claim H and execution of `verify` NLI auditing.
- **Turn 3**: Final `conclude_search` tool invocation. Early termination is triggered if two consecutive verification failures occur.

Algorithm 1: Harness State Transition & NLI Verification Workflow

Data: Board SGF Sequence $\mathcal{S}$, KJD Trie-S Index $\mathcal{T}$, NLI Model $\mathcal{M}_{NLI}$
Output: Audited Working Memory Cache $S_t = (H_t, C_t, D_t, V_t)$
$S_0 \leftarrow$ InitializeState()
Turn 1: Execute TrieRetrieval$(\mathcal{S}, \mathcal{T}) \rightarrow C_t, D_t$
Turn 2: Sensei Agent Formulates Claim $H =$ SynthesizeClaim(C_t)
$P \leftarrow$ ExtractLinearStoryPremise(D_t)
$(Label, Score) \leftarrow \mathcal{M}_{NLI}(P, H)$
if $Label ==$ 'entailment' **then**
    $V_t \leftarrow$ 'VERIFIED_TRUE'
else
    $V_t \leftarrow$ 'WARN_UNGROUNDED'
Turn 3: Execute ConcludeSearch() → Lock Cache for Lecturer

4.2 Specialized Multi-Agent Collaboration

We divide labor across three specialized agents powered by `Qwen3.5-27B`:

1. **Go Sensei Agent**: Handles state machine execution, 4-mirror rotation normalization, and Trie-S RAG retrieval verification without generation overhead.
2. **Lecturer Agent**: Consolidates audited RAG contexts, GTP board features, and KataGo win-rate metrics to synthesize structured English tactical commentary.
3. **Translator Agent**: Translates English commentary into professional, localized Traditional Chinese, enforcing strict domain terminology alignment.

© 2026 Information Processing Society of Japan

5 Experiments and Empirical Results

5.1 Experimental Setup

Experiments were conducted on a workstation featuring an AMD Ryzen 7 9700X CPU, 64GB RAM, and an NVIDIA RTX 5090 GPU (32GB VRAM). The backbone LLM was `Qwen3.5-27B-Q4_K_M.gguf` executed via `llama-server` with a 32,768 context window. Evaluation was performed on 10 representative opening test cases (`bjjq_1`, `bjjq_2`, `bujushili_1` – `6`, `tu226`, `tu227`).

5.2 LLM-as-a-Judge Protocol & Metrics

Following the LLM-as-a-Judge paradigm [6], outputs were evaluated independently by dual frontier models (Claude 3.5 Sonnet and Gemini 1.5 Pro) across five 5-point Likert scale metrics against ground-truth human grandmaster commentaries ($\mathcal{C}_{gold}$):

- M_1 **(Physical Accuracy)**: Move coordinates, colors, and quadrant orientations.
- M_2 **(Tactical Alignment)**: Professional terminology usage.
- M_3 **(Strategic Analysis)**: Whole-board judgment and KataGo win-rate/score lead interpretation.
- M_4 **(Fact Consistency)**: Agreement with reference joseki variations.
- M_5 **(Readability & Localization)**: Fluency and pedagogical clarity.

5.3 7-Stage Progressive Ablation Study

To systematically evaluate the individual contribution of each architectural component, we implemented a 7-stage progressive ablation study:

- **Step 1 (Raw SGF Baseline)**: Direct SGF prompt to Naive LLM.
- **Step 2 (+ 2D Features & KataGo)**: Adding spatial GTP tags and KataGo win-rate/score evaluations.
- **Step 3 (+ Standard Vector RAG)**: Adding standard dense vector similarity retrieval.
- **Step 4 (+ Trie-S RAG)**: Replacing standard RAG with hierarchical Trie-S node retrieval.
- **Step 5 (Single-Agent Full-Pipeline)**: Single LLM handling tool calling, RAG, and generation.
- **Step 6 (Multi-Agent w/o NLI Verification)**: Multi-Agent setup with NLI verification disabled.
- **Step 7 (Proposed Full System)**: Multi-Agent pipeline with full state machine, Trie-S RAG, KataGo, and NLI verification.

Table 1 7-Stage Progressive Ablation Study Results (1–5 Likert Scale)

Ablation Step	M_1	M_2	M_3	M_4	M_5	Overall Mean
Step 1 (Raw SGF)	2.00	2.10	2.00	2.00	2.00	**2.02**
Step 2 (+ 2D & KataGo)	4.40	4.10	3.70	3.70	3.90	**3.96**
Step 3 (+ Standard RAG)	2.90	2.70	2.50	2.60	2.50	**2.64**
Step 4 (+ Trie-S RAG)	4.10	4.20	3.70	3.90	3.80	**3.94**
Step 5 (Single Agent)	4.50	4.10	3.60	3.50	3.80	**3.90**
Step 6 (w/o NLI)	1.30	1.40	1.30	1.30	1.10	**1.28**
Step 7 (Proposed Full)	**4.80**	**4.70**	**4.10**	**4.20**	**4.70**	**4.50**

5.4 Detailed Empirical Breakdown of Ablation Steps

1. **Impact of 2D Features (Step 1 vs. Step 2)**: Adding 2D spatial tags and KataGo metrics causes M_1 (Physical Accuracy) to jump from 2.00 to 4.40, lifting overall score from 2.02 to 3.96.

2. **Standard RAG Chunk Pollution (Step 2 vs. Step 3)**: Introducing standard vector RAG causes a sharp drop in overall score to 2.64. Dense similarity matching retrieves off-board joseki chunks (e.g., retrieving Star point variations for Komoku moves), corrupting prompt context.

3. **Precision Recovery via Trie-S RAG (Step 3 vs. Step 4)**: Replacing vector RAG with Trie-S RAG recovers overall performance to 3.94. Enforcing exact DFS tree node matching eliminates off-board retrieval pollution.

4. **Single-Agent Task Overload (Step 4 vs. Step 5)**: A single LLM handling tool calling, retrieval, and text generation exhibits prompt context overload, leading to early truncation and slightly lower overall consistency (3.90).

5. **Catastrophic Context Collapse (Step 5 vs. Step 6)**: Disabling NLI hypothesis verification in Step 6 results in a catastrophic score collapse down to 1.28 overall. Unaudited RAG context induces runaway text generation and ungrounded claims.

6. **Synergistic Peak Performance (Step 7)**: The proposed full system achieves an overall score of **4.50**, confirming that state-machine control, Trie-S RAG, KataGo numerical analysis, and NLI verification operate synergistically.

5.4.1 Summary of Ablation Findings

In summary, the 7-stage ablation study reveals three core takeaways: (1) 2D spatial feature extraction is indispensable for eliminating spatial hallucinations (M_1 : $2.00 \rightarrow 4.40$); (2) exact Trie-S RAG topological matching is mandatory to prevent dense vector retrieval pollution ($2.64 \rightarrow 3.94$); and (3) NLI hypothesis verification acts as an essential safety gate, preventing unaudited RAG claims from inducing catastrophic prompt context collapse ($1.28 \rightarrow 4.50$).

© 2026 Information Processing Society of Japan

5.5 Token Consumption and Latency Speedup

As shown in Table 2, our multi-agent framework reduces inference latency from 178.6s to 86.1s (a 2.07× speedup / 51.8% reduction) compared to Naive LLM in thinking mode, while maintaining a virtually identical total token budget (12,483 vs. 12,530 tokens).

Table 2 Token consumption and latency benchmark comparison

(a) Naive LLM Baseline (Thinking Mode)

Metric	Value
Prompt / Completion Tokens	329 / 12,201
Total Tokens / Latency	12,530 / 178.6s

(b) Proposed Harness-RAG System

Metric	Value
Prompt / Completion Tokens	7,510 / 4,973
Total Tokens / Latency	12,483 / 86.1s

(c) Performance Comparison & Speedup

Metric	Performance Comparison
Prompt / Completion Delta	$+7,181$ / $-7,228$
Total Token / Latency Delta	-47 (-0.38%) / -92.5s ($2.07\times$)

5.6 Qualitative Analysis & Output Failure Mode Baseline

Naive SGF Baseline Output Failure Mode (Step 1): Direct ingestion of raw 1D SGF sequences causes the naive LLM to get trapped in reasoning loops within internal thinking tags (`<think>`), leading to token exhaustion, reasoning tag entrapment, and output collapse (generating empty or single-sentence truncated responses).

5.7 Qualitative Output Evolution of Case (`bujushili_1`)

To demonstrate the concrete qualitative progression across ablation steps, we examine the output evolution for case `bujushili_1`:

Step 1: Naive SGF Baseline Output Snapshot (`bujushili_1`)

Output Analysis (1,433 chars): Direct SGF prompt. Completely lacks KataGo numerical win-rate data, lacks RAG reference grounding, and exhibits step-number/coordinate hallucinations.

Step 3: Standard Dense RAG Output Snapshot (`bujushili_1`)

Output Analysis (Truncated to 197 chars): Vector similarity matching retrieves off-board joseki chunks, polluting prompt context and causing severe output truncation.

Step 6: Multi-Agent w/o NLI Verification Snapshot (`bujushili_1`)

Output Analysis (Catastrophic Expansion to 8,974 chars): Unaudited RAG context induces prompt instruction leakage, runaway text generation, and system failure.

Step 7: Proposed Full System Complete Verbatim Output (`bujushili_1`)

 # 專業開局覆盤報告
1. 佈局架構與戰略主題
本局的開局序列展現了經典的**小目對星位**動態。黑 1（R16）在右上角佔據穩固的小目，優先考慮角地與勢力。白 2（D16）則在左上角以星位回應，顯示出對大場架構與行棋速度的偏好。
棋局迅速演變為**二連星開局**的結構。白 4（D4）在左下角佈下二連星陣勢，而黑 3（Q3）則穩住右下角的小目。黑 5（P17）在右上角進行守角，旨在鞏固黑 1（R16）所建立的角地。這形成了一盤均衡的棋局，黑方專注於角部的穩定性，白方則試圖在左側構築勢力。
2. 核心定式變化與戰術決策點
最關鍵的局部變化發生在右下角。- **掛角**：白 6（Q5）對黑 3（Q3）進行高掛。- **應對**：黑 7（R5）以碰回應，這是銳利的戰術選擇。白 8（R6）扳，黑 9（R4）長以維持棋形。白 10（Q7）形成虎口，確保角地。- **後續**：黑 11（O3）下一間跳，將勢力向外延伸。
在標準定式變化中，黑 7（R5）之後，白方本可考慮其他應對以避免棋形凝重，但白 8（R6）的扳是有效的積極後續手段。
棋局中盤，左側戰事活躍。黑 15（C14）對白 2（D16）進行低掛。白 16（F17）以低位守角回應，鞏固左上角的架構。左下角則展開複雜交換，始於黑 17（C6）低掛與白 18（F3）低位守角。黑 19（B4）小飛與白 20（C3）尖形成緊湊的局部戰鬥，最終由黑 21（E6）一間跳解決。
進入官子階段，黑 25（P15）在右上角走出狗頭形，意在削減白方潛力或鞏固地盤。白 26（J17）與黑 27（L17）代表最後的全局調整，其中黑 27（L17）為常規落子，旨在平衡全局。
3. AI 評估與全局判斷
根據 **AI 評估矩陣**，黑方目前佔據顯著優勢，**勝率為 75.78%，領先分數為 1.50 目**。- **戰略平衡**：高勝率顯示，黑方穩固的小目戰略（黑 1、黑 3）結合精準的守角開局（黑 5），在領地效率上優於白方重視架構的打法。- **關鍵時刻**：右下角定式（第 6 至 11 手）似乎被黑方正確處理，阻止了白方在該角獲得過多勢力。AI 傾向於黑方將角部轉化為確定領地，同時保持足夠勢力以對抗白方的二連星佈局。- **全局判斷**：儘管白方在左側積極行棋（第 15 至 21 手），黑方持續的常規落子與穩固棋形（例如黑 25 狗頭形）維繫了領先優勢。1.50 目的領先代表局面舒適但尚未決出勝負，需要謹慎的官子行棋將優勢轉化為勝利。
結論：黑方成功應對開局，利用小目的靈活性確保領地，並有效中和了白方的架構企圖。目前的棋局狀態有利於黑方，前提是中盤階段不發生重大戰術失誤。

6 Discussion & Scope Limitations

While corner joseki choices depend on surrounding stones, our system handles whole-board position via

© 2026 Information Processing Society of Japan

three complementary layers: (1) 4-mirror rotation normalization, (2) KataGo full 19×19 board win-rate evaluations, and (3) Sabaki pattern tags for global side-splitting (Fentou) and framework (Moyo) boundaries.

We explicitly define the scope of this study to **Go Opening Commentary Generation**. Mid-game tactical battles and endgame counting involve complex liberty races (Semeai) and Tesuji calculations that require specialized tree-search expansion, which we leave for future work.

7 Conclusion

This study presented a state-driven multi-agent Go opening commentary framework combining a 2D Go Board Analyzer, Trie-S RAG, KataGo GTP integration, NLI hypothesis verification, and rigid Harness state machine control. Evaluated across a 7-stage progressive ablation study, our system achieved a top score of 4.50 out of 5.00 on human grandmaster alignment benchmarks while delivering a $2.07\times$ inference speedup.

Acknowledgement

The authors would like to thank anonymous referees for their valuable comments in improving the overall quality of this paper. This work was supported in part by the National Science and Technology Council under contracts NSTC 113-2221-E-259-018-MY3 and NSTC 115-2622-E-259-001.

References

[1] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] D. J. Wu, "Accelerating self-play learning in Go," *arXiv preprint arXiv:1902.10565*, 2019.

[3] P. Lewis, E. Perez, A. Piktus, *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Conference on Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 9459–9474, 2020.

[4] G. Odom and A. Dinerchtein, "Kogo's Joseki Dictionary." http://waterfire.us/joseki.htm, 2026.

[5] P. Jiang, Z. Shi, K. Hong, *et al.*, "Harness-1: Reinforcement learning for search agents with state-externalizing harnesses," *arXiv preprint arXiv:2606.02373*, 2026.

[6] L. Zheng, W.-L. Chiang, Y. Sheng, *et al.*, "Judging LLM-as-a-judge with MT-bench and chatbot arena," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, pp. 46599–46623, 2023.

© 2026 Information Processing Society of Japan