NBA Fan AI Agent: A Natural Language Understanding System for NBA Fans

Kevin Zhou
yrzhou@umich.edu

Jiadong Zhu
jiadongz@umich.edu

[GitHub Repository]

Abstract

We present a domain-specific Natural Language Understanding (NLU) system for NBA information queries that performs intent classification and slot filling, then directly retrieves answers from the Ball Don't Lie API. We compare three NLU approaches: a rule-based method, a fine-tuned BERT model, and a zero-shot LLM (Qwen3). Evaluation reveals a clear accuracy-speed trade-off, with the LLM achieving highest accuracy but slowest inference, the rule-based method being fastest but least accurate, and BERT+spaCy providing a balance. Results demonstrate that domain-specific NLU can achieve strong performance with limited training data, with method selection depending on accuracy vs. speed priorities.

1 Introduction

As NBA fans, we often spend significant time searching online for game information and player statistics. Older voice assistants like Siri were limited to basic queries such as scores and match times, while newer systems like Apple Intelligence and ChatGPT rely on retrieving information from websites, making responses slower and less seamless. For fans who want quick, accurate, and user-friendly access to NBA-related data, these solutions remain less than ideal.

This project addresses this gap by building a domain-specific NBA Fan AI Agent that understands natural language questions about basketball and directly retrieves answers from the Ball Don't Lie API. The system performs intent classification and slot filling to extract structured information from queries, then uses entity linking and API integration to return accurate, real-time responses.

While intent classification and slot filling have been extensively studied in NLU literature, most existing systems are either general-purpose (requiring large training datasets) or rely on external web retrieval (introducing latency). Our approach differs by focusing specifically on the NBA domain, integrating directly with a structured API, and comparing three NLU methods: a rule-based baseline, a fine-tuned BERT model, and a zero-shot LLM approach.

This work demonstrates how domain-specific NLU can achieve strong performance with limited data while maintaining low latency, benefiting basketball fans, sports media developers, and analytics platforms. Our main contributions are: (1) showing that domain-specific NLU can work well with small datasets, (2) presenting an end-to-end system with fast API integration, and (3) providing a comparative analysis of rule-based, fine-tuned, and zero-shot approaches for domain-specific NLU.

2 Data

Our dataset for the NBA Fan AI Agent consists of 300 annotated queries covering NBA player and team information. The dataset was generated using AI to create natural-language question templates with varied phrasings.

The dataset supports two intent classes: `player_info` (200 examples) and `team_info` (100 examples). We define 15 attributes total: 10 player attributes (`position`, `height`, `weight`, `jersey_number`, `college`, `country`, `draft_year`, `draft_round`, `draft_number`, `team`) and 5 team attributes (`conference`, `division`, `city`, `full_name`, `abbreviation`). Each attribute has 20 examples with varied phrasings, ensuring linguistic diversity while maintaining consistent semantic meaning.

Each example is stored as a JSON record. The training and validation sets use a `<name>` placeholder for entity names (player or team names), which allows the model to learn intent and attribute patterns independently of specific entities. For example:

```
{"text": "Which college did <name> play
for?",
"intent": "player_info",
"slots": {"attribute": "college"} }
```

The dataset is split into training, validation, and test sets using an 80-10-10 ratio. The training and validation sets (240 examples) use the <name> placeholder format and are used for fine-tuning the BERT model. The test set (30 examples) contains actual entity names (e.g., "Jordan Poole", "Lakers") to enable evaluation of entity extraction capabilities across all NLU methods. For example, the same query appears in the test set as:

```
{"text":  "Which  college  did  Jordan
Poole play for?",
"intent": "player_info",
"slots":    {"attribute":    "college",
"entity": "Jordan Poole"} }
```

Because all examples were AI-generated, the intent and slot annotations are consistent and noise-free, providing reliable supervision for model training and evaluation.

3 Related Work

The tasks of intent classification and slot filling have been extensively studied in the literature on natural language understanding (NLU), particularly within the context of spoken dialogue systems. Early work often treated the two tasks separately, but more recent approaches emphasize joint modeling for improved performance. Here, we highlight several representative papers that inform our project.

Benchmark datasets such as ATIS, Snips, and CLINC150 have become the standard for evaluating intent classification and slot filling models. ATIS focuses on airline travel queries and remains a long-standing benchmark, Snips covers diverse consumer domains such as music and weather, and CLINC150 provides 150 intents across domains along with out-of-scope queries for OOD evaluation. These datasets can inform the design of our NBA-specific dataset by illustrating how to balance intent coverage, slot schemas, and robustness to domain variation.

Guo et al. (2014) introduced recursive neural networks (RecNNs) for joint intent classification and slot filling on the ATIS corpus. Their model leveraged compositional parse-tree structures to encode hierarchical semantics, outperforming traditional CRF pipelines. However, their approach relied on syntactic parses and domain-specific grammars that do not generalize well to informal sports queries.

Goo et al. (2018) proposed the slot-gated mechanism that explicitly links intent prediction and slot filling via shared attention weights. Evaluated on ATIS and Snips, it achieved notable gains in joint accuracy and demonstrated that cross-task information flow improves robustness to noisy input. This concept directly informs our architecture: NBA queries often contain ambiguous mentions (e.g., "Heat" vs. "hot") that benefit from intent-slot interaction.

Chen et al. (2019) fine-tuned BERT for joint NLU, achieving state-of-the-art F1 scores on multiple benchmarks. Their results confirmed that pretrained contextual encoders capture slot boundaries and rare entities better than sequence-tagging baselines. We adopt this insight by training a multi-task BERT model specialized for NBA data, extending their method to a new, domain-specific dataset.

Qian and Yu (2019) explored meta-learning for domain adaptation in dialogue systems, allowing rapid transfer to new tasks with few labeled examples. Their meta-training strategy motivates our inclusion of out-of-domain (OOD) evaluation, ensuring that our NBA model can gracefully reject unrelated inputs.

Finally, Larson et al. (2019) introduced CLINC150, a large-scale benchmark with 150 intents and OOD detection tests. Their evaluation highlighted the importance of balanced data and explicit OOD categories. We follow their principles in constructing our dataset to maintain intent balance and incorporate unseen query types for realistic evaluation.

Together, these studies reveal that (1) joint models outperform independent ones, (2) pretrained transformers enhance low-resource generalization, and (3) domain adaptation remains critical. Our project builds on these lessons but extends them to a sports-specific NLU setting—a domain with unique linguistic ambiguity (team names, player aliases) and dynamic data requirements. By designing a compact, well-annotated dataset and a lightweight multi-task BERT model, we aim to demonstrate that high-accuracy NLU is achievable even in narrow, rapidly evolving domains such as professional basketball.

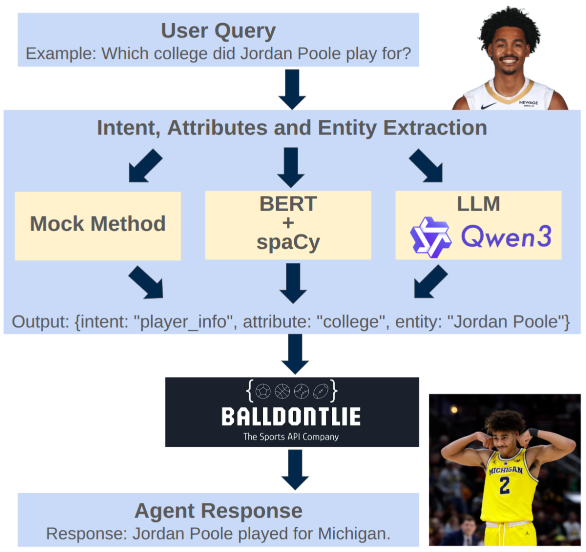
User Query
Example: Which college did Jordan Poole play for?
Intent, Attributes and Entity Extraction
Mock Method
BERT + spaCy
LLM
Qwen3
Output: {intent: "player_info", attribute: "college", entity: "Jordan Poole"}
BALLDONTLIE
The Sports API Company
Agent Response
Response: Jordan Poole played for Michigan.

Figure 1: System flow

4 Methods

4.1 System Overview

Our NBA Fan AI Agent follows an end-to-end pipeline that processes natural language queries through four main stages: (1) natural language understanding (NLU) to extract intent, attributes, and entities, (2) entity linking to map names to API IDs, (3) API integration to retrieve structured data, and (4) response formatting to generate natural language answers. Figure 1 illustrates the complete workflow.

The system supports three distinct NLU approaches for comparison: a rule-based mock method for fast baseline performance, a fine-tuned BERT model for efficient inference, and a zero-shot LLM approach using Qwen3-4B-Instruct (Yang et al., 2025) for flexible generalization. All three methods produce the same structured output format (intent, attribute, entity), enabling seamless integration with the downstream API connection components. The modular design allows each NLU method to be evaluated independently while sharing the same entity linking, API routing, and response formatting modules.

4.2 Mock Method

The mock method serves as a simple rule-based baseline that uses keyword matching and regular expressions to predict intent, attribute, and entity names without any machine learning. This approach provides a fast, interpretable baseline for comparison and enables testing of the API integration pipeline before training more sophisticated models.

The mock predictor operates on raw text input, converting queries to lowercase and searching for predefined keywords and patterns. It uses hard-coded lists of team names, player names, and attribute keywords, along with regular expressions to match intent patterns. The prediction algorithm determines intent by matching patterns or entity keywords, extracts attributes by searching keyword mappings, and extracts entities by matching team or player keywords with alias handling. The implementation uses Python's built-in `re` module, requiring no external NLP libraries. We chose this approach because it requires no training data, provides immediate interpretability, and enables rapid prototyping. The design prioritizes speed and simplicity over accuracy.

4.3 BERT Method

The BERT method uses a fine-tuned multi-task learning model based on `bert-base-uncased` to jointly predict intent and attribute classifications. Entity extraction is handled separately using spaCy's transformer-based named entity recognition model.

Training and validation examples use the `<name>` placeholder format. Text is tokenized with `BertTokenizer` (max length 64), and labels are encoded using `LabelEncoder`. The dataset is split by intent-attribute combination: 16 examples per combination for training, 2 for validation, and 2 for testing.

The model (`BertForIntentAndAttr`) extends BERT with two linear classification heads for intent (2 classes) and attribute (15 classes). The 768-dimensional pooled output from BERT's `[CLS]` token serves as the shared representation, passed through dropout ($p = 0.2$) and fully connected layers. The model jointly optimizes:

$$\mathcal{L} = \mathcal{L}_{intent} + \mathcal{L}_{attribute}.$$

During inference, entities are replaced with `<name>` using spaCy NER to match the training distribution. Entity extraction uses spaCy's `en_core_web_trf` model, identifying player names as `PERSON` and team names as `ORG` or `GPE`.

Training uses PyTorch with AdamW optimizer (learning rate 2×10^{-5}, batch size 16, 15 epochs) and linear warm-up. We chose multi-task learning to leverage shared representations, and `<name>` placeholders enable generalization to any player or team name.

4.4 LLM Method

We used the Qwen3-4B-Instruct-2507-FP8 model for zero-shot prediction of intent, attribute, and entity extraction without any fine-tuning. This approach leverages the model's instruction-following capabilities through prompt engineering.

The input query is passed directly to the model without preprocessing. The model uses its chat template to format the prompt, which includes the user query, task instructions, and output format specifications. The prompt explicitly defines valid intents and attributes, groups attributes by intent, and instructs the model to extract the entity name. The prompt template is structured as follows:

```
Question: {user_query}

Classify this NBA question by
identifying intent, attribute, and
entity.

CRITICAL RULES - MUST FOLLOW:
1. If the question asks about a
PLAYER, intent MUST be "player_info"
and attribute MUST be one of: college,
country, draft_number, draft_round,
draft_year, height, jersey_number,
position, team, weight
2. If the question asks about a TEAM,
intent MUST be "team_info" and attribute
MUST be one of: abbreviation, city,
conference, division, full_name
3. These combinations are FIXED - you
CANNOT mix them
4. Extract the player or team name
mentioned in the question

Respond ONLY with valid JSON: {"intent":
"...", "attribute": "...", "entity":
"..."}
```

The Qwen3-4B-Instruct model is a 4-billion parameter instruction-tuned language model with FP8 quantization. It is loaded with `device_map="auto"` and uses float16 precision on GPU. Generation parameters include `max_new_tokens=128` and `temperature=0.1` for consistent outputs. The model's response is parsed using regular expressions to extract JSON, with fallback mechanisms to handle malformed outputs and invalid intent-attribute combinations.

The implementation uses `transformers` for model loading and generation, `torch` for tensor operations, and Python's built-in `json` and `re` modules for response parsing. We chose zero-shot prediction to avoid fine-tuning costs and enable rapid adaptation to new query types.

4.5 API Connection

The API connection module bridges NLU predictions with the Ball Don't Lie API (Ball Don't Lie, 2024), handling entity linking, intent routing, and response formatting. This component enables the system to retrieve real NBA data and return natural language responses.

After NLU prediction, the system receives intent and slots (containing entity name and attribute). The `EntityLinker` class maps entity names to API IDs using a multi-stage approach: (1) checking a hardcoded dictionary of team aliases, (2) exact matching against cached data, (3) partial matching, and (4) fuzzy matching using `difflib.SequenceMatcher` with similarity thresholds of 0.7 for teams and 0.75 for players. Teams and players are cached on first access, with players limited to 2,000 results.

The `APIRouter` class routes predicted intents to appropriate API handlers. For `team_info` and `player_info` intents, it validates slots, links entity names to IDs, calls the API service, and extracts the requested attribute. The API returns structured JSON data, which the `ResponseFormatter` class converts into natural language using intent-specific templates and handling edge cases such as missing data.

The implementation uses the `balldontlie` Python SDK (Ball Don't Lie, 2024) for API access and `difflib` for fuzzy string matching. We chose fuzzy matching to handle variations in entity names, and the multi-stage approach prioritizes speed while maintaining robustness. Caching reduces API latency, and error handling ensures graceful degradation when entities cannot be found or API calls fail.

5 Evaluation and Results

5.1 Evaluation Setup

We evaluated all three NLU methods on a held-out test set of 30 examples, which contains actual entity names (player and team names) to enable evaluation of entity extraction capabilities. The test set maintains the same distribution as the training data: 20 player-related queries and 10 team-related queries, with 2 examples per attribute across all 15 attributes.

For each method, we report performance across three subtasks: (1) *intent classification*—predicting whether a query is about player information or team information, (2) *attribute classifica-*

Method	Overall	Intent	Attribute	Entity	Time (ms)
Mock	66.67%	86.67%	66.67%	66.67%	0.18
BERT+spaCy	73.33%	83.33%	80.00%	73.33%	47.15
LLM (Qwen3)	**96.67%**	**100%**	**100%**	**96.67%**	1,573.16

Table 1: Performance and inference time comparison across all three NLU methods on the test set (30 examples). Overall accuracy requires all three tasks (intent, attribute, entity) to be correct. Times are averaged over 30 test examples.

tion—identifying which specific attribute is being queried (e.g., college, height, conference), and (3) *entity extraction*—extracting the player or team name from the query. We use accuracy as the primary metric for all three tasks, defined as the fraction of examples where all three components (intent, attribute, and entity) are predicted correctly. We also report per-task accuracies and inference time to understand the trade-offs between methods.

5.2 Results

Table 1 summarizes the performance of all three methods on the test set. The mock method, which uses keyword matching and regular expressions without any machine learning, achieves 66.67% overall accuracy (all three tasks correct) with an average inference time of 0.18 milliseconds. This demonstrates that simple rule-based approaches can achieve reasonable performance for straightforward queries but struggle with linguistic variations and edge cases.

The BERT+spaCy method achieves 73.33% overall accuracy with an average inference time of 47.15 milliseconds. Breaking down the performance by task, BERT achieves 83.33% accuracy on intent classification, 80.00% accuracy on attribute classification, and 73.33% accuracy on entity extraction. The model performs well on most attributes but struggles with certain paraphrases, particularly for player position queries (0% recall) and some draft-related attributes. Entity extraction using spaCy's NER model achieves 73.33% accuracy, with common failure modes including possessive forms (e.g., "Anthony Davis's" $\rightarrow$ "Anthony Davis") and queries where the entity name is not recognized as a PERSON or ORG entity.

The LLM method (Qwen3-4B-Instruct) achieves the highest performance with 96.67% overall accuracy, correctly predicting intent, attribute, and entity for 29 out of 30 examples. The method achieves perfect accuracy (100%) on both intent and attribute classification, demonstrating the model's strong instruction-following capabilities. Entity extraction achieves 96.67% accuracy, with only one error where the model extracted "Miami" instead of "Heat" for a query about the Miami Heat's city. However, this superior accuracy comes at a significant computational cost, with an average inference time of 1,573.16 milliseconds—over 30 times slower than BERT and over 8,700 times slower than the mock method.

The mock method is nearly instantaneous, making it suitable for applications requiring sub-millisecond response times. BERT+spaCy provides a good balance between accuracy and speed, suitable for real-time applications. The LLM method, while achieving the highest accuracy, requires over 1.5 seconds per query, making it less suitable for interactive applications but potentially valuable for batch processing or when accuracy is paramount.

5.3 Error Analysis

The mock method's errors primarily stem from limited keyword coverage and inability to handle paraphrases. For example, it fails to recognize "alma mater" as equivalent to "college" and struggles with draft-related queries (draft number, round, year) that don't contain explicit keywords. The method also has difficulty extracting full names when only partial names appear in queries.

BERT+spaCy's errors reveal limitations in handling certain linguistic patterns. The model incorrectly classifies some player queries as team queries (e.g., "What's LeBron James's alma mater?" $\rightarrow$ team_info), suggesting that the training data may not have sufficient coverage of possessive forms and alternative phrasings. The model also struggles with the "position" attribute, achieving 0% recall, likely because position queries use varied phrasings (e.g., "Can you give me Kyrie Irving's position?") that differ from the training templates. Entity extraction errors often involve possessive forms (e.g., "Giannis Antetokounmpo's" $\rightarrow$ "Giannis Antetokounmpo's") or cases where spaCy fails to recognize entity boundaries.

The LLM method's single error demonstrates its robustness to linguistic variation, but also reveals a limitation: when asked “Which city is home to the Heat?”, the model extracted “Miami” (the answer) instead of “Heat” (the entity name). This suggests the model may sometimes confuse the entity being queried with the answer to the query, though this occurred only once in our test set.

6 Discussion

Our evaluation reveals a clear accuracy-speed trade-off across the three NLU approaches, demonstrating that method selection depends critically on application requirements. The results show that domain-specific NLU can achieve strong performance with limited training data, but the choice between accuracy and speed requires careful consideration.

6.1 Overall Performance Assessment

The LLM achieves the highest accuracy but is over 30 times slower than BERT+spaCy, making it unsuitable for real-time dialogue applications. BERT+spaCy balances accuracy and speed, making it the most practical choice for production systems requiring real-time responses. The rule-based Mock Method, while fastest, achieves lower accuracy, limiting its utility to rapid prototyping. For an end-user application, BERT+spaCy's accuracy means approximately one in four queries may fail, which may be acceptable for some use cases but requires improvement for production deployment. The sub-50ms inference time demonstrates BERT+spaCy's suitability for real-time dialogue.

6.2 Comparison with Baselines

The Mock Method's performance, while higher than random chance, reveals the limitations of rule-based approaches. The method fails on linguistic variation, ambiguous queries, and edge cases not covered by its keyword patterns. This confirms that understanding query semantics requires learning semantic relationships rather than simple pattern matching. The gap between Mock and BERT+spaCy demonstrates that even with limited training data, learned representations capture patterns that rule-based systems miss. The further improvement with the LLM suggests that larger models with more parameters can better handle the complexity of natural language queries, but at significant computational cost.

6.3 Why Each Approach Performed as It Did

The Mock Method's performance reflects its keyword-based design—it succeeds on queries matching its patterns but fails on linguistic variation, synonyms, and paraphrases. The BERT+spaCy pipeline's moderate accuracy stems from learned semantic representations, with the `<name>` placeholder strategy enabling generalization to unseen entities and multi-task learning leveraging shared representations. However, the error rate suggests that 300 training examples may be insufficient to capture all linguistic patterns. The LLM's high accuracy demonstrates the power of large pre-trained models, which can handle diverse query patterns through zero-shot prompting, but the high latency reflects the computational overhead of token-by-token generation.

6.4 Error Patterns and Design Implications

The error rate in BERT+spaCy likely stems from insufficient training data, the separation of entity extraction from classification (losing context), and the use of general-purpose NER for sports-specific recognition. The LLM's low error rate suggests that with sufficient model capacity, most queries can be handled correctly, but the speed penalty is prohibitive for real-time applications. The Mock Method's errors cluster around queries with linguistic variation or patterns not in its keyword dictionary. These patterns suggest that each approach has complementary strengths, potentially motivating ensemble methods or hybrid approaches in future work.

6.5 Trade-offs Between Approaches

The results demonstrate a clear accuracy-speed trade-off. The Mock Method prioritizes speed at the cost of accuracy, suitable for rapid prototyping. BERT+spaCy offers a balanced middle ground for production systems requiring real-time responses. The LLM prioritizes accuracy at the cost of speed, suitable for batch processing. This comparison helps practitioners choose methods based on their constraints: speed-critical applications (Mock), balanced production systems (BERT+spaCy), or accuracy-critical batch processing (LLM).

6.6 Limitations and Context

Our evaluation on synthetic queries enables controlled experimentation but may not reflect real-world performance. Natural queries contain typos, abbreviations, slang, and complex phrasings absent from our templates. The two-intent limitation (player_info, team_info) restricts applicability—real fans ask about games, statistics, trades, and other complex queries requiring multi-hop reasoning. These limitations suggest that while our approach is promising, real-world deployment requires expanded coverage and evaluation on natural queries.

7 Conclusion

This work demonstrates that domain-specific NLU systems can achieve strong performance with limited training data, with method selection depending on whether accuracy or speed is prioritized. Our main contributions are threefold. First, we show that domain-specific NLU works effectively with small datasets when using appropriate training strategies—specifically, the <name> placeholder approach enables models to learn semantic patterns independently of specific entities. Second, we present an end-to-end system integrating NLU with structured API access, eliminating slow web retrieval. Third, we provide a comparative analysis of three NLU approaches, revealing a clear accuracy-speed trade-off that helps practitioners choose methods based on their constraints.

Our results demonstrate a clear trade-off: the LLM achieves the highest accuracy but is slowest, the rule-based Mock Method is fastest but least accurate, and BERT+spaCy balances both. The trade-off shows that practitioners must choose based on their priorities: speed-critical applications benefit from rule-based methods, balanced production systems benefit from fine-tuned models, and accuracy-critical batch processing benefits from large language models. BERT+spaCy offers a practical middle ground suitable for real-time dialogue applications.

The principles and architecture generalize beyond NBA to other domain-specific dialogue systems requiring fast, accurate information retrieval from structured APIs. The combination of template-based training, multi-stage entity linking, and direct API integration provides a blueprint for building efficient NLU systems in specialized domains. Future work should address entity extraction limitations, expand intent coverage, and evaluate on natural user queries.

8 Other Things We Tried

During development, we considered several alternative approaches that we ultimately did not implement.

8.1 BERT Entity Extraction

For the BERT method, we initially attempted to train BERT for entity extraction in addition to intent and attribute classification. However, the model struggled to generalize to unseen entity names, required larger datasets with entity span annotations, and performed worse than spaCy's specialized NER model. We switched to the current BERT+spaCy approach, where BERT handles classification while spaCy handles entity extraction.

8.2 ChatGPT API for LLM Method

For the LLM method, we considered using OpenAI's ChatGPT API instead of deploying Qwen3-4B-Instruct locally. However, we decided against it due to network latency, per-query costs, and the desire to maintain consistency with our other local methods. Using a local model also provides more control and eliminates API rate limit concerns.

9 Future Work and Reflections

Based on our results, the most promising next steps are: (1) improving entity extraction through domain-specific NER models or joint training to address the 73.33% accuracy bottleneck, (2) expanding to game_info intent to increase system utility and cover more user queries, (3) building an interactive web UI to enable real-world user testing and natural query collection, (4) implementing enhanced error handling for edge cases discovered during API integration, and (5) conducting comprehensive evaluation on natural user queries to assess real-world performance and identify additional failure modes.

The perfect classification accuracy suggests that the <name> placeholder approach and multi-task learning are effective strategies that could be extended to additional intents. The entity extraction bottleneck clearly indicates that investing in domain-specific entity recognition would yield significant improvements. The fast inference time (29.78 ms) demonstrates the system's suitability for real-time applications, making it a viable foundation for building a production system. These results point toward a clear path for future development: focus on entity extraction improvements while maintaining the speed and classification accuracy we've achieved.

10 Group Effort

This project was a collaborative effort between two team members. We found that dataset creation took longer than expected, primarily due to balancing between attribute coverage and natural phrasing diversity. However, this early investment proved valuable, as it simplified model training and debugging later. Moving forward, we allocated more time for evaluation and integration, since combining NLU output with live API responses introduces additional edge cases (e.g., player name variants, missing data).

The division of labor remains balanced:

Team Member 1 (Kevin Zhou):

- Responsible for fine-tuning and evaluating the BERT model
- Responsible for the LLM baseline (Qwen3-4B-Instruct zero-shot approach)
- Conducted hyperparameter tuning and optimization
- Performed statistical analysis of results
- Implemented the BERT multi-task model architecture
- Developed the training pipeline and preprocessing utilities

Team Member 2 (Jiadong Zhu):

- Conducted integration testing with the Ball Don't Lie API
- Performed system latency evaluation and performance benchmarking
- Implemented the rule-based mock predictor for baseline comparison
- Created the end-to-end pipeline orchestration

Collaborative Work:

Both team members collaborated on:

- Dataset validation and quality assurance
- Evaluation metrics design (intent accuracy, macro F1, slot F1, entity extraction accuracy, latency, and cost)
- Error analysis and failure mode identification
- Poster preparation and presentation
- Final report writing

This ensures balanced contributions to the core NLP tasks, while still allowing complementary specialization in different approaches. The modular architecture of the system enabled parallel development, with each member focusing on different components while maintaining clear interfaces for integration. Regular code reviews and joint testing sessions ensured consistency and quality throughout the project.

References

Ball Don't Lie. 2024. Ball don't lie api. Accessed: 2024.

Qian Chen, Zhu Zhuo, and Wen Wang. 2019. Bert for joint intent classification and slot filling. *Preprint*, arXiv:1902.10909.

Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 2018. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 753–757, New Orleans, Louisiana. Association for Computational Linguistics.

Daniel Guo, Gokhan Tur, Wen-tau Yih, and Geoffrey Zweig. 2014. Joint semantic utterance classification and slot filling with recursive neural networks. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 554–559.

Stefan Larson, Anish Mahendran, Joseph J. Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K. Kummerfeld, Kevin Leach, Michael A. Laurenzano, Lingjia Tang, and Jason Mars. 2019. An evaluation dataset for intent classification and out-of-scope prediction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Kun Qian and Zhou Yu. 2019. Domain adaptive dialog generation via meta learning. *Preprint*, arXiv:1906.03520.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.