УДК 551.46.077:629.584
EDN: UNZGPZ
DOI: 10.37102/1992-4429_2026_55_01_06

НЕЙРОННАЯ СЕТЬ ДЛЯ ГЕНЕРАЦИИ ПРОГРАММ-МИССИЙ АНПА

А.С. Пугачев, А.И. Боровик

Статья посвящена решению задачи автоматизации составления программ-миссий для автономных необитаемых подводных аппаратов (АНПА) с использованием нейронной сети, работающей без необходимости использования сети Интернет. В работе рассматривается полный цикл дообучения предобученной большой языковой модели архитектуры «трансформер» Llama 3.1 8B Instruct для генерации программ-миссий на языке подводных исследований – ЯППИ, начиная от формирования обучающей выборки и заканчивания инференсом модели. Представлены и описаны методы LORA и 4-битное квантование для повышения скорости обучения языковой модели и уменьшения требований к устройствам, используемым для запуска обученной модели. Описано применение RAG-файлов, содержащих актуальную информацию по командам языка (их синтаксис, семантика, обязательные и необязательные параметры), а также релевантные данные при составлении миссий для конкретных акваторий. В результате работы была получена локально работающая нейронная сеть, показывающая высокую точность генерации кода на ЯППИ на основе запросов на естественном (русском) языке, а также хорошую устойчивость к шуму во входных данных.

Ключевые слова: АНПА, нейронные сети, миссия, интеллектуальная система помощи оператору АНПА, язык программирования миссий АНПА, ЯППИ.

Введение

Автономные необитаемые подводные аппараты (АНПА) стали незаменимым инструментом для решения широкого круга задач: от инспекции подводной инфраструктуры и подводных поисковых операций до экологического мониторинга и научных исследований Мирового океана. Ключевым преимуществом АНПА является их способность работать по заранее заданной программе-миссии без наличия устойчивого канала связи между аппаратом и судном-носителем и прямого контроля аппарата со стороны оператора [1, 2].

Одним из главных этапов применения АНПА является процесс составления и отладки программы-миссии аппарата перед его запуском [3, 4]. В каждом аппарате (или семействе аппаратов) используется собственный механизм задания миссий – графические утилиты для их визуального составления на карте местности либо специализированные языки программирования со своим синтаксисом, семантикой, специфическими командами и особенностями. Создание корректной миссии при этом обычно требует не только детального знания языка либо соответствующих графических утилит, но также и специфики работы бортовой системы управления, навигационных средств аппарата, его технических характеристик, особенностей конкретной акватории и т.п. Все это формирует высокий порог входа для новых операторов АНПА, создает дефицит квалифицированных кадров в отрасли, а для самих специалистов – сильную когнитивную нагрузку во время экспедиционных работ. При этом в настоящее время в связи с быстрым развитием больших языковых моделей (Large Language Models, LLM) на базе нейронных сетей последние все чаще применяются в качестве интеллектуальных помощников в узкоспециализированных областях деятельности – программировании, медицине, юриспруденции, науке и т.д. Одним из перспективных направлений применения LLM видится автоматизация узкоспециализированного программирования, например, разработка миссий для подводных роботов.

В данной статье рассматривается автоматизация разработки программ для АНПА с использованием дообученной большой языковой модели. В качестве базовой модели была выбрана Llama 3.1 8B Instruct [5], показавшая высокую эффективность в задачах

следования инструкциям при локальной работе (без подключения к сети Интернет) на компьютере умеренной производительности [6]. В статье описывается полный цикл внедрения модели: от ее загрузки и подготовки обучающей выборки до обучения и последующего использования для генерации кода. Также рассмотрена интеграция RAG для повышения точности генерации.

1. Архитектура и используемые технологии

Модель Llama 3.1 8B Instruct – это одна из последних LLM серии Meta Llama, предназначенная для диалоговых систем и следования инструкциям. Она содержит около 8 миллиардов параметров и обладает поддержкой длинных контекстов [5]. Данная языковая модель была внедрена в нейросетевой модуль разрабатываемой в ИПМТ интеллектуальной системы помощи оператору АНПА по результатам проведённого ранее сравнительного анализа нейронных сетей при решении задачи генерации миссий [6].

Архитектура нейросетевого модуля состоит из пяти базовых слоев – слой конфигурации, слой загрузки модели, слой загрузки данных, слой обучения и слой инференса. Диаграмма компонентов нейросетевого модуля представлена на рис. 1, описание слоев – в табл. 1.

Таблица 1. Слои системы

Слой системы	Назначение
Слой конфигурации	Предоставляет доступ к параметрам обучения и инференса нейросети
Слой загрузки данных	Загружает обучающую выборку из JSON-файлов и дополнительный контекст из внешних источников
Слой загрузки модели	Проверяет наличие локальных файлов LLM и при необходимости загружает их с HuggingFace Hub.
Слой обучения	Токенезирует текстовые данные, подающиеся на вход, настраивает гиперпараметры для дообучения, применяет механизмы оптимизации обучения и экономии памяти
Слой инференса	Хранит историю диалогов с пользователем, генерирует код миссий и осуществляет валидацию сгенерированного контента

Разработка модуля велась на языке программирования Python, версия 3.13, использованные библиотеки перечислены в табл. 2.

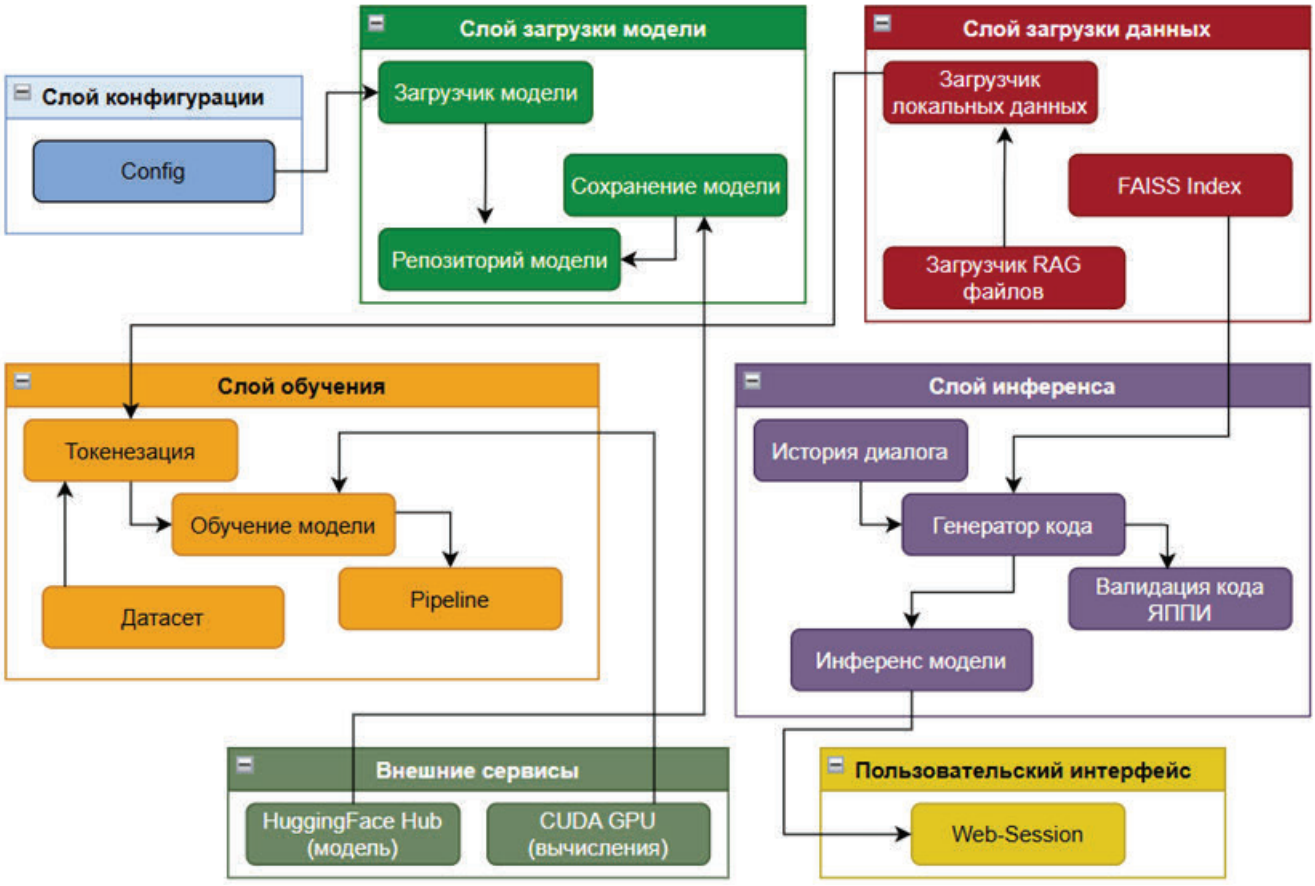

Рис. 1. Диаграмма интеллектуальной системы помощи оператору АНПА

Таблица 2. Использованные при разработке модули Python

Библиотека	Описание
transformers	Загрузка и работа с нейросетями архитектуры «трансформер»
peft	Настройка нейросети через LoRA
bitsandbytes	Квантование весов нейросети
datasets	Обработка выборок (датасетов)
langchain, faiss, langchain-huggingface	Интеграция RAG и построение системы поиска
huggingface_hub	Авторизация для работы с открытыми (open-weight) нейросетями

2. Подготовка обучающей выборки

Для проведения сравнительного анализа нейронных сетей при решении задачи генерации миссий ранее была сформирована первичная обучающая выборка, состоящая из 200 пар вида «запрос пользователя – код миссии на ЯППИ». ЯППИ – язык программирования подводных исследований – универсальный язык для описания миссий АНПА, позволяющий создавать человекочитаемые миссии для подводных роботов, который был разработан в рамках решения этой же задачи ранее [7]. При создании первичной выборки использовались только материалы реальных миссий, накопленные институтом за долгое время работы с разными аппаратами при решении разнообразных задач – от поиска затонувших объектов до обследования трубопроводов. Текстовые описания к миссиям задавались на основе операторских журналов и заметок, для генерации кода на ЯППИ использовались автоматические трансляторы из использующихся в проектах института языков описания миссий (в настоящий момент в разных аппаратах используется 4 разных подхода).

Подобного набора данных может быть достаточно для проведения сравнительного анализа, но недостаточно для финального обучения интеллектуального помощника – выборка покрывает почти все возможные варианты задач АНПА, но далеко не все возможные пользовательские запросы, которые могут включать специфический сленг, идиоматические выражения, транслитерации терминов и т.п. По этой причине было решено расширить обучающую выборку за счет генерации различных вариантов текстовых запросов («промтов») для исходного набора миссий обучающей выборки. Для автоматизации данной работы были использованы мощные языковые модели, доступные онлайн – Qwen 3 от Alibaba и Grok от xAI. Для каждой миссии из исходного набора с помощью данных языковых моделей на основе исходного операторского описания было сформировано 15 разновидностей запросов на естественном языке, содержащих сленговые и идиоматические выражения. Результат генерации нейросетей после был поверхностно проверен операторами для удаления из него запросов, содержащих «галлюцинации» искусственного интеллекта.

Полученная в результате обучающая выборка содержит 3000 пар «запрос пользователя – код на ЯППИ» с 200 уникальными миссиями АНПА, покрывающими практически все варианты промышленного и научного использования подводных роботов. Фрагмент полученной выборки представлен на рис. 2, поле с меткой “text” содержит описание задачи на естественном языке, “labels” – код миссии на ЯППИ.

3. Подготовка модели к обучению

Обучение больших языковых моделей требует значительных вычислительных ресурсов – обычно

```
{
"text": "Проведи обследование прямоугольной акватории с координатами левой нижней вершины: 131.911296 восточной долготы и
43.107656 северной широты. Длина участка - 800 метров, ширина - 500 метров. Использовать ГБО, траектория - вертикальный меандр,
межгалсовое расстояние - 80 метров, высота движения - 10 метров, скорость - 1 м/с. Средняя глубина места составляет 100 метров.",
"labels": "миссия(глубина_места(100)) обследование_фигуры(фигура(прямоугольник), координаты(131.911296*ВД, 43.107656*СШ),
длина(800), ширина(500), межагалс(80), высота(10), прибор(гбо), траектория(меандр, вертикально), скорость(1))"
},
{
"text": "Проведи осмотр трубопровода змейкой. Координаты трубы: [131.912345, 43.210987], [131.915678, 43.210876], [131.918901,
43.210765]. Используй МЛЭ в высокочастотном режиме. Высота - 7 метров, скорость - 0.8м/с. Таймаут связи - 20 минут. Максимальная
глубина - 50 метров.",
"labels": "миссия(глубина_места(50), максимальное_время(1200))
обследование_линии(траектория(ломанная), координаты([131.912345*ВД, 43.210987*СШ], [131.915678*ВД, 43.210876*СШ], [131.918901*ВД,
43.210765*СШ]), скорость(0.8), высота(10), прибор(млэ, ВЧ))"
},
{
"text": "Выполни фотосъемку точки с координтами 138.9012 ВД и 50.1234 СШ по траектории меандр. Параметры: высота 3м, скорость 0.4
м/с. Рельеф местности ровный, глубина до 40 метров."
"labels": "миссия(глубина_места(40)) обследование_точки(траектория(меандр), координаты(138.9012*ВД, 50.1234*СШ), высота(3),
скорость(0.4), прибор(фотокамера))"
}
```

Рис. 2. Фрагмент обучающей выборки

используются суперкомпьютеры либо машины с несколькими параллельно работающими высокопроизводительными видеокартами или специальными нейронными сопроцессорами (NPU) [8]. Дообучение разрабатываемой нейросети выполнялось на относительно «среднем» по вычислительной мощности компьютере с процессором Intel Core i7-11800H (2.3 GHz, 8 ядер), 16 Гб оперативной памяти типа DDR-4 и видеокартой NVIDIA GeForce RTX 3070 (8 ГБ видеопамяти). Чтобы оптимизировать использование доступных ресурсов и сделать обучение возможным на такой машине, были использованы специальные методы подготовки модели – *4-битное квантование* и *адаптация через LoRA* (Low-Rank Adaptation). 4-битное квантование с nf4 позволяет загрузить модель в память, значительно сокращая её объём, а настройка LoRA даёт возможность эффективно обучать модель, изменяя лишь небольшое количество её параметров [9, 10].

Первый этап обучения модели – ее загрузка в оперативную память видеокарты (или компьютера, если обучение проводится без использования видеокарты). Для уменьшения потребления видеопамяти при загрузке модели использовалось нормальнораспределённое 4-битное квантование, реализованное в python-библиотеке bitsandbytes (режим «nf4»). Настройки, задаваемые в классе BitsAndBytesConfig, приведены на рис. 3. Без квантования обучение LLM типа Llama-3.1-8B-Instruct (имеющей ~8 миллиардов параметров) с использованием всего одной графической карты с 8 Гб видеопамяти заняло бы очень много времени.

```
if self.device == "cuda":
    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=torch.float16,
        bnb_4bit_use_double_quant=True,
    )
```

Рис. 3. Настройки 4-битного квантования модели через класс BitsAndBytesConfig

Второй этап – перевод модели из режима «инференса» в режим обучения, который корректно настраивает компоненты модели для выполнения обратного распространения ошибки и обновления весов. Для этой задачи использовалась функция «*prepare_model_for_kbit_training*» из Python библиотеки «*peft*».

На финальном подготовительном этапе была выполнена настройка модели методом LoRA, что позволило не переобучать все параметры модели, а лишь

```
lora_config = LoraConfig(
    r=8,
    lora_alpha=32,
    target_modules=target_modules,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)
```

Рис. 4. Конфигурация LoRA

подстроить некоторые из них. Конфигурация LoRA представлена на рис. 4.

- Параметр «r» задаёт ранг низкоранговой декомпозиции, используемой в каждом модифицируемом слое. Чем меньше его значение, тем меньше обучаемых параметров добавляется при адаптации модели. В данном случае выбрано значение 8.
- Параметр «*lora_alpha*» определяет масштаб, с которым LoRA-адаптация применяется к исходным весам модели. Его увеличение усиливает вклад обучаемых весов, что улучшает качество адаптации, но при малом объёме обучающих данных повышает риск переобучения.
- Параметр «*target_modules*» задаёт список типов модулей, к которым применяется LoRA.
- Параметр «*lora_dropout*» определяет долю выходных активаций слоя LoRA, которые случайным образом обнуляются на каждом шаге обучения для предотвращения переобучения.
- Значение «*bias=none*» означает, что смещения не обучаются, что немного уменьшает число настраиваемых параметров и снижает требования к оперативной памяти.

4. Процесс обучения

Обучение модели начинается с загрузки предварительно квантованной версии Llama 3.1 8B Instruct. Если локальная обученная модель уже существует, она загружается из директории models, в противном случае инициируется процесс обучения с нуля. Для обучения используется оптимизатор «*paged_adamw_8bit*», который также помогает снизить количество используемой памяти. Самым важным параметром является размер пакета (batch size), который установлен 1 из-за ограничений видеопамяти. Для компенсации малого размера пакета применяется накопление градиентов (gradient accumulation) с ша-

гом 4, позволяя имитировать больший пакет без увеличения нагрузки на память.

В процессе обучения модель учится преобразовывать естественно-языковые описания подводных миссий в код на ЯППИ, следуя заданным правилам. Например, все команды должны начинаться с конкретных ключевых слов («*миссия*», «*обследование_фигуры*», «*обследование_точки*», «*обследование_линии*» и так далее), а параметры указываться в строгом скобочном формате [7].

После завершения обучения модель сохраняется в указанную директорию. Важным этапом является объединение адаптированных параметров LoRA с основной моделью и их последующая сериализация, что позволяет в дальнейшем использовать модель без дополнительных зависимостей.

5. Интеграция RAG

Механизм *Retrieval-Augmented Generation* (RAG) позволяет нейронной сети получать доступ к внешним знаниям. В нашем случае RAG используется для:

- дополнительной настройки нейросети для строгого следования к специфическому синтаксису;
- уточнения технических параметров и возможностей конкретного аппарата;
- уточнения характеристик конкретной акватории – максимальная глубина места, возможное отстояние от грунта и т. п.;
- любых иных специфических настроек, учет которых может повлиять на создаваемые миссии.

RAG интегрирован в конвейер генерации кода следующим образом: при получении пользовательского запроса система сначала ищет наиболее релевантные фрагменты текста из базы знаний, а затем использует их как дополнительный контекст для модели. Это позволяет уточнять параметры команд, избегать ошибок в синтаксисе и учитывать специфические требования миссии. Также в дальнейшем предполагается дополнять директорию с RAG-файлами таким образом, чтобы нейросеть могла самостоятельно достраивать программу-миссию, самостоятельно заполняя не введенные пользователем данные.

Документы RAG хранятся в отдельной директории и загружаются при помощи класса DirectoryLoader из библиотеки langchain.

Поскольку исходные документы могут быть большими, их необходимо разбить на фрагменты оптимального размера для эффективного поиска. Для этого используется *CharacterTextSplitter* с параметрами:

- chunk_size=500 – максимальная длина фрагмента в символах;
- chunk_overlap=50 – перекрытие между соседними фрагментами для сохранения контекста.

После разбиения система проверяет, что полученные фрагменты не пусты, и выводит их для изучения.

Для быстрого поиска релевантных фрагментов используется векторное представление текстов. С помощью модели *sentence-transformers/all-MiniLM-L6-v2* фрагменты преобразуются в числовые векторы (эмбеддинги), которые затем индексируются в FAISS – библиотеке для приближенного поиска ближайших соседей [11].

Процесс включает:

- инициализацию модели эмбеддингов – выбирается легковесная, но достаточно точная модель для баланса между скоростью и качеством;
- создание индекса – FAISS группирует векторы так, чтобы похожие фрагменты находились рядом в пространстве признаков;
- сохранение индекса в памяти – после построения индекс остается активным для быстрого доступа во время генерации кода;
- если индекс успешно создан, система уведомляет об этом и переходит к этапу генерации. В случае ошибки (например, из-за нехватки памяти) RAG временно отключается.

Интеграция RAG улучшает качество генерации кода на ЯППИ, позволяя модели учитывать актуальную базу знаний, предупредить пользователя о возможных проблемах при выполнении миссии, автоматически заполнить некоторые обязательные параметры функций ЯППИ (например, глубину места в районе работ).

6. Генерация кода

Качество генерации кода во многом зависит от того, насколько точно нейросеть сможет интерпретировать намерения пользователя. Чтобы повысить точность этой интерпретации, входной запрос расширяется контекстом: к нему добавляются системные правила, релевантные фрагменты из внешних источников (RAG), а также история диалога. Эту задачу выполняет функция «*_build_prompt*», которая формирует единый, структурированный промпт для последующей генерации.

В качестве системных правил в настоящее время используется набор правил генерации кода на ЯППИ, представленный на рис. 5.

```
Правила синтаксиса ЯППИ:

ОБЯЗАТЕЛЬНЫЕ ПРАВИЛА:
1. Все параметры должны состоять из строчных букв.
2. Должны указываться только те параметры, которые были явно заданы пользователем.
3. На выходе только чистый код на ЯППИ.
4. Используй строгую скобочную нотацию: команда(параметр1(значение), параметр2(значение)).
```

Рис. 5. Системные правила генерации кода

Поддержка контекста диалога с нейросетью реализуется следующим образом: внутри системы история диалога хранится в специальном буфере, ограниченном 10 последними взаимодействиями. При каждом новом обращении в запрос автоматически включаются как формулировки предыдущих запросов, так и соответствующий им код на ЯППИ. В таком случае модель может использовать ранее заданные пользователем параметры, а также корректно обрабатывать пользовательские отсылки к предыдущим запросам.

В любой момент допускается сбросить накопленный контекст командой «*clear*». В этом случае история очищается и система начинает новую сессию планирования без влияния предыдущих инструкций

Сформированный промпт используется в функции «*_run_inference*», которая вызывает метод «*generate*» класса «*AutoModelForCausalLM*», выполняющий генерацию текста с заданными параметрами (рис. 6).

В табл. 3 описаны параметры, передаваемые в функцию «*generate*».

В ходе выполнения тестирования разработанного нейросетевого модуля было произведено исследование зависимости количества ошибок генерации от значения параметра «*температура*». На низких значениях (0.1–0.3) модель выдавала шаблонные ответы – выставляла одинаковое время миссии независимо от задачи или обрезала вывод. С ростом температуры ситуация менялась: начиная с 0.5 появлялись лишние команды, а к 0.7 модель уже генерировала недопустимые конструкции или пустые значения параметров (рисунки 7, 8).

Зависимость точности генерации вывода от температуры можно рассчитать по выведенной эмпирически формуле:

$$score = 100 - (x \cdot 0,4 + y \cdot 0,3 + z \cdot 0,2 + p \cdot 0,1), \quad (1)$$

где –x количество скрытых синтаксических ошибок, при большом количестве которых трансляция кода станет невозможна; y – количество избыточных команд, использование которых не предполагалось пользовательским запросом; z – количество параметров, заданных пользователем, но проигнорированных моделью; p – шаблонные значения – значения параметров функций, «придуманные» нейросетью.

```python
def _run_inference(self, prompt: str) -> str:
    try:
        inputs = self.tokenizer(prompt, return_tensors="pt").to(self.device)

        with torch.no_grad():
            outputs = self.model.generate(
                **inputs,
                max_new_tokens=Config.MAX_NEW_TOKENS,
                temperature=Config.TEMPERATURE,
                top_p=Config.TOP_P,
                repetition_penalty=getattr(Config, 'REPETITION_PENALTY', 1.1),
                pad_token_id=self.tokenizer.eos_token_id,
                eos_token_id=self.tokenizer.eos_token_id,
                use_cache=True,
            )

        generated_ids = outputs[0][len(inputs["input_ids"][0]):]
        result = self.tokenizer.decode(generated_ids, skip_special_tokens=True)
        return result
    except Exception as e:
        raise RuntimeError(f"Ошибка при генерации на {self.device}: {str(e)}")
```

Рис. 6. Прототип функции _run_inference

Таблица 3. Параметры вывода языковой модели

max_new_tokens	Максимальное количество токенов в сгенерированном тексте
temperature	Температура, определяющая уникальность вывода
top_p	Распределение вероятности распространенных токенов, определяющее вывод каждого следующего слова
repetition_penalty	Штраф за повторение текста
use_cache	Использование кэш Key-Value для оптимизации
pad_token_id	Токен заполнения, использующийся для выравнивания последовательностей разной длины до одного размера
eos_token_id	Токен, обозначающий конец последовательности

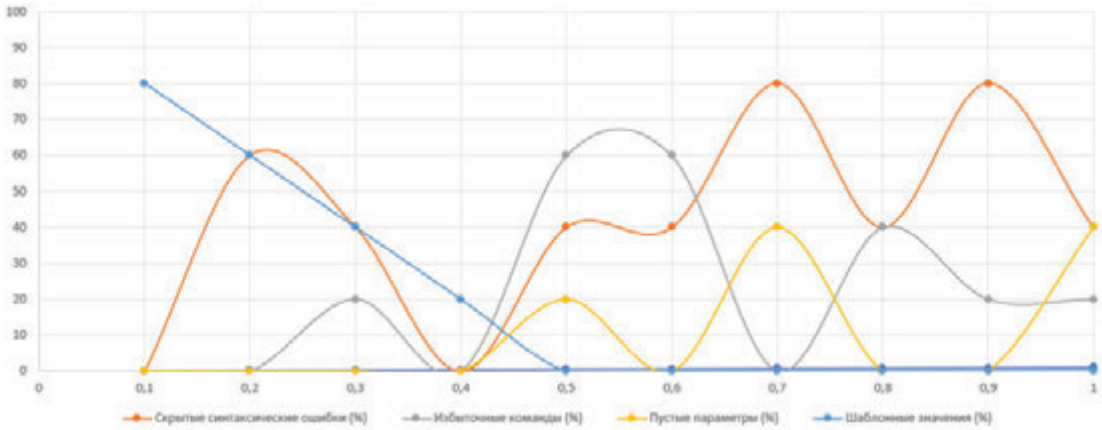

Рис. 7. График зависимости ошибок вывода от температуры

Как видно по данному тестированию, оптимальным оказалось значение температуры 0.4 – оно и используется в системе по умолчанию.

При генерации текста также используется параметр *top-p* (0.9), который ограничивает множество возможных следующих токенов только наиболее вероятными вариантами с целью избегания потенциально ошибочных конструкций [12].

Максимальное количество слов в одной генерации (миссии) установлено на уровне 600. Данное значение было определено на основе анализа обучающего датасета – в нем не нашлось миссий длиннее 300 знаков.

После генерации моделью исходного кода миссии выполняется его постобработка, направленная на приведение результата к корректному синтаксису языка ЯППИ. Система отбирает только те строки, которые начинаются с разрешённых команд – таких как «*миссия*», «*обследование_фигуры*», «*обследование_линии*», «*обследование_точки*», «*движение*», «*зависание*» и так далее. Любые пояснения, комментарии или свободные формулировки, не соответствующие формату, игнорируются.

Если ни одна строка не соответствует ожидаемой структуре, активируется резервный механизм: система пытается выделить хотя бы фрагменты, содержащие скобочную запись вида «*()*». Далее применяется автоматическая коррекция типичных ошибок. Например, удаляются недопустимые параметры и исправляется возможное дублирование. Дополнительно проводится нормализация пробелов и запятых внутри скобок.

7. Системные требования

Процессы дообучения и инференса предъявляют существенно разные требования к вычислительным

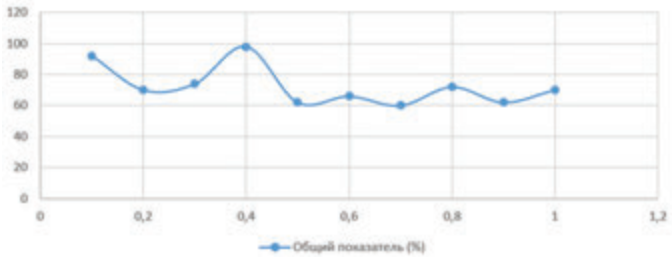

Рис. 8. Общая оценка зависимости ошибок вывода от температуры

ресурсам. Дообучение требует значительно бóльших объёмов видеопамяти (VRAM), оперативной памяти и вычислительной мощности, поскольку включает градиентные обновления миллионов параметров модели и работу с большими последовательностями данных. В связи с этим дообучение возможно только на GPU, так как центральные процессоры (CPU) не обеспечивают достаточной производительности для работы с современными языковыми моделями.

В то же время генерация кода на основе уже обученной модели может выполняться как на GPU, так и на CPU, что дает возможность запускать нейросеть даже на устройствах без дискретной графики, хотя с заметным снижением скорости генерации.

В табл. 4 представлены минимальные системные требования для тренировки модели. В табл. 5 представлены минимальные системные требования, позволяющие запустить нейросеть для генерации кода на ЯППИ.

Таблица 4. Системные требования для обучения модели

Параметр	Минимальное значение	Рекомендуемое значение
GPU	NVIDIA RTX 3070 (8 ГБ VRAM)	NVIDIA RTX 4080 (16+ ГБ VRAM)
RAM	16 Гб	32 Гб
CPU	8 ядер	12 ядер

Таблица 5. Системные требования для инференса модели

Режим	Параметр	Минимальное значение	Рекомендуемое значение
GPU-режим	GPU	NVIDIA RTX с не менее 6 ГБ VRAM	NVIDIA RTX 3070 (8 ГБ VRAM)
	RAM	Не менее 8 Гб	16 Гб
	CPU	Не менее 4 ядер	6 ядер
CPU-режим	GPU	–	–
	RAM	Не менее 8 Гб	16 Гб
	CPU	Не менее 6 ядер	8 ядер

Результаты

В результате дообучения модель безошибочно генерирует ЯППИ-код миссий АНПА на основе русскоязычных запросов в свободной форме. На рис. 9 приведен пример запроса на составление миссии по обследованию акватории прямоугольной формы, в котором намеренно пропущены некоторые обязательные параметры миссии: «*время без связи*», «*глубина местности*», «*максимальное время работы*», также в запросе не указаны высота работы аппарата и режим (фигура) обследования. Как видно, нейросеть смогла подобрать корректные значения для режима обследования («*вертикальный меандр*») и высоты работы (*10 метров*) исходя из используемого в миссии исследовательского оборудования – гидролокатора бокового обзора (ГБО). В то же время в существующих RAG-файлах модель не смогла найти значения для пропущенных параметров команды «*миссия*», поскольку в запросе не определен аппарат, для которого создается миссия, а также координаты района обследования не могут быть однозначно соотнесены с заданными в RAG-файлах акваториями. По этой причине модель не стала «придумывать» значения для пропущенных параметров, а просто не указала их. Как видно на приведенном скриншоте, при дальнейшей обработке сгенерированного кода интерпретатор языка ЯППИ обнаружил отсутствие обязательных параметров и запросил их у пользователя. В целом можно сказать, что в показанном примере все механизмы интеллектуальной системы помощи отработали как задумано.

Следует отметить, что модель в текущем состоянии также справляется и с генерацией кода сложных миссий, содержащих условия, например миссий, в которых осуществляется поиск объектов по ГБО с их дальнейшим фотообследованием – в этих случаях генерируется корректный код на ЯППИ с описанием соответствующих событий [7].

Заключение

Была сформирована обучающая выборка, содержащая 3000 пар вида «запрос пользователя – код на ЯППИ». С использованием данной выборки и разработанного механизма было выполнено дообучение предобученной большой языковой модели Llama 3.1. 8B-instruct. Получившаяся в результате нейросеть позволяет генерировать эффективный код миссий для автономного необитаемого подводного аппарата на ЯППИ на основе запросов на естественном языке. Разработанная нейросеть используется в качестве центрального компонента нейросетевого модуля разрабатываемой в настоящий момент интеллектуальной системы помощи оператору АНПА.

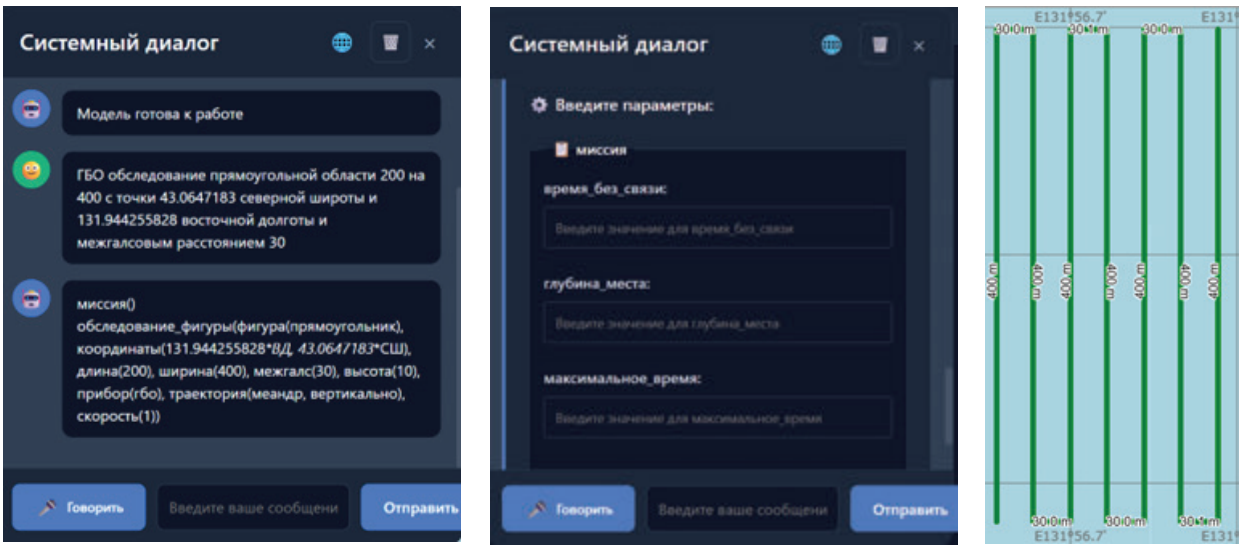

Рис. 9. Результат работы нейросети после обучения

Дальнейшие работы

В настоящий момент проводится полноценное тестирование модели на объемной тестовой выборке (также созданной на основе реальных операторских запросов и генераций больших языков моделей, доступных онлайн). Результаты данного тестирования позволят получить количественные метрики качества генерации кода разработанной нейросетью, а также оценить удобство использования интеллектуальной системы помощи операторами и будут опубликованы в дальнейших работах.

Одной из ключевых возможностей разрабатываемой системы должно стать постоянное обучение на основе реального взаимодействия с операторами в ходе выполнения экспедиционных задач. Для этого планируется реализовать механизм *коллективного дообучения модели*: система будет локально сохранять всю историю диалогов пользователя с нейросетью во время экспедиции, а при подключении к компьютерной сети Института – автоматически отправлять эти данные в централизованное хранилище, где они должны загружаться в единый обучающий датасет. В дальнейшем все локальные версии модели при подключении к сети должны будут автоматически дообучаться на основе измененного общего датасета, что позволит организовать эффективный обмен опытом между экспедициями с АНПА ИПМТ ДВО РАН.

СПИСОК ИСТОЧНИКОВ

1. Агеев М.Д., Касаткин Б.А., Рылов Н.И. и др. Автоматические подводные аппараты. Л.: Судостроение, 1981. 224 с.
2. Инзарцев А.В., Киселев Л.В., Костенко В.В., Матвиенко Ю.В., Павин А.М., Щербатюк А.Ф. Подводные робототехнические комплексы: системы технологии применения. Владивосток: ДВО РАН ИПМТ, 2018. 368 с.
3. Инзарцев А.В. К вопросу о способах представления задания для обследовательского подводного робота / А.В. Инзарцев, А.В. Багницкий // Технические проблемы освоения Мирового океана. 2013. Т. 5. С. 376–381. EDN YSWCTR.
4. Багницкий А.В., Инзарцев А.В. Автоматизация подготовки миссии для автономного необитаемого подводного аппарата в задачах обследования акваторий // Подводные исследования и робототехника. 2010. № 2(10). С. 17–24.
5. HuggingFace Hub. meta-llama/Llama-3.1-8B-Instruct. URL: https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct.
6. Пугачев А.С., Боровик А.И. Сравнительный анализ нейронных сетей трансформеров при решении задачи генерации миссий АНПА. Подводные исследования и робототехника. 2025. №3(53). С. 58–63. DOI: 10.37102/1992-4429_2025_53_03_06. EDN: VQOQQZ.
7. Пугачев А.С. ЯППИ – универсальный язык Программирования миссий АНПА / А.С. Пугачев, А.И. Боровик // Подводные исследования и робототехника. 2024. № 3(49). С. 26–37. doi: 10.37102/1992-4429_2024_49_03_03. EDN GVSAMY.
8. Zhang Z.X., Wen Y.B., Lyu H.Q.,et al. AI Computing Systems for Large Language Models Training // J. Comput. Sci. Technol. 2025. Vol. 40. P. 6–41. https://doi.org/10.1007/s11390-024-4178-1.
9. Hu E.J., Shen Y., Wallis P., Allen-Zhu Z., Li Y., Wang S., Wang L., Chen W. LoRA: Low-Rank Adaptation of Large Language Models. arXiv. https://doi.org/10.48550/arXiv.2106.09685. -2021
10. Liu S., Liu Z., Huang X., Dong P., Cheng K.-T. LLM-FP4: 4-Bit Floating-Point Quantized Transformers. arXiv. https://doi.org/10.48550/arXiv.2310.16836. -2023.
11. HugginFace Hub. sentence-transformers/all-MiniLM-L6-v2. [Электронный ресурс]. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.
12. Holtzman A., Buys J., Du L., Forbes M., Choi Y. The Curious Case of Neural Text Degeneration. URL: https://arxiv.org/abs/1904.09751?spm=a2ty_o01.29997173.0.0.1f0d5171Q1vJzw&file=1904.09751.ASd

Сведения об авторах:

ПУГАЧЕВ Андрей Сергеевич, аспирант, младший научный сотрудник лаборатории робототехнических систем
Федеральное государственное бюджетное учреждение науки Институт проблем морских технологий им. академика М.Д. Агеева Дальневосточного отделения Российской академии наук Лаборатория робототехнических систем, инженер
Адрес: 690091, Владивосток, ул. Суханова, 5А
Область научных интересов: вычислительные системы, компиляторы, нейронные сети, прикладное программное обеспечение
E-mail: pugachevas@marine.febras.ru
Тел.: 8(423)2-215-545, доб. 509

БОРОВИК Алексей Игоревич, кандидат технических наук, ведущий научный сотрудник, заведующий лабораторией робототехнических систем
Федеральное государственное бюджетное учреждение науки Институт проблем морских технологий им. академика М.Д. Агеева Дальневосточного отделения Российской академии наук
Адрес: 690091, Владивосток, ул. Суханова, 5А
Область научных интересов: АНПА, ТНПА, системы управления роботами, робототехнические программные платформы, разработка программного обеспечения
E-mail: alexey@borovik.me
Тел.: 8(423)2-215-545, доб. 510
ORCID: 0000-0002-9696-2751

Для цитирования:

Пугачев А.С., Боровик А.И. НЕЙРОННАЯ СЕТЬ ДЛЯ ГЕНЕРАЦИИ ПРОГРАММ-МИССИЙ АНПА // Подводные исследования и робототехника. 2026. № 1 (55). С. 59–68. DOI: 10.37102/1992-4429_2026_55_01_06. EDN: UNZGPZ.

DOI: 10.37102/1992-4429_2026_55_01_06

NEURAL NETWORK FOR GENERATING AUV MISSIONS

A.S. Pugachev, A.I. Borovik

The article addresses the problem of automating the generation of mission programs for autonomous underwater vehicles (AUVs) using a neural network, working without Internet access. It examines the full cycle of fine-tuning a pre-trained large language model Llama 3.1 8B Instruct with the «transformer» architecture for generating mission programs in a specialized underwater research programming language (URPL), starting from dataset formation and ending with model inference. LORA and 4-bit quantization methods are presented and described for increasing the speed of language model training and reducing the requirements for devices used to run the trained model. The use of RAG files containing up-to-date information on language commands (syntax, semantics, mandatory and optional parameters), as well as relevant data for generating missions for specific water areas, is described. The result of the work was a neural network that demonstrated high accuracy in generating code in URPL based on queries in natural (Russian) language, as well as good resistance to noise in the input data.

Keywords: AUV, neural networks, mission, intelligent AUV-operator assistance system, AUV mission programming language, URPL.

References

1. Ageev M.D., Kasatkin B.A., Rylov N.I. i dr. Avtomaticheskie podvodnye apparaty. L.: Sudostroenie, 1981. 224 s. (in Russ.)

2. Inzarcev A.V., Kiselev L.V., Kostenko V.V., Matvienko Yu.V., Pavin A.M., Shcherbatyuk A.F. Podvodnye robototekhnicheskie kompleksy: sistemy tekhnologii primeneniya. Vladivostok: IPMT DVO RAN, 2018. 368 s. (in Russ.)

3. Inzarcev A.V. K voprosu o sposobah predstavleniya zadaniya dlya obsledovatel'skogo podvodnogo robota / A.V. Inzarcev, A.V. Bagnickij // Tekhnicheskie problemy osvoeniya Mirovogo okeana. 2013. Vol. 5. P. 376–381. EDN: YSWCTR. (in Russ.)

4. Bagnickij A.V., Inzarcev A.V. Avtomatizaciya podgotovki missii dlya avtonomnogo neobitaemogo podvodnogo apparata v zadachah obsledovaniya akvatorij // Podvodnye issledovaniya i robototekhnika. 2010. No. 2(10). S. 17–24. (in Russ.)

5. HuggingFace Hub. meta-llama/Llama-3.1-8B-Instruct. URL: https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct.

6. Pugachev A.S., Borovik A.I. Sravnitel'nyj analiz nejronnyh setej transformerov pri reshenii zadachi generacii missij ANPA. Podvodnye issledovaniya i robototekhnika. 2025. No. 3 (53). P. 58–63. DOI: 10.37102/1992-4429_2025_53_03_06. EDN: VQOQQZ. (in Russ.)

7. Pugachev A.S. YaPPI – universal'nyj yazyk Programmirovaniya missij ANPA / A.S. Pugachev, A.I. Borovik // Podvodnye issledovaniya i robototekhnika. 2024. No. 3(49). S. 26–37. doi: 10.37102/1992-4429_2024_49_03_03. EDN GVSAMY. (in Russ.)

8. Zhang Z.X., Wen Y.B., Lyu H.Q.,et al. AI Computing Systems for Large Language Models Training // J. Comput. Sci. Technol. 2025. Vol. 40. P. 6–41. https://doi.org/10.1007/s11390-024-4178-1.

9. Hu E.J., Shen Y., Wallis P., Allen-Zhu Z., Li Y., Wang S., Wang L., Chen W. LoRA: Low-Rank Adaptation of Large Language Models. arXiv. https://doi.org/10.48550/arXiv.2106.09685. -2021

10. Liu S., Liu Z., Huang X., Dong P., Cheng K.-T. LLM-FP4: 4-Bit Floating-Point Quantized Transformers. arXiv. https://doi.org/10.48550/arXiv.2310.16836. -2023.

11. HugginFace Hub. sentence-transformers/all-MiniLM-L6-v2. URL: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

12. Holtzman A., Buys J., Du L., Forbes M., Choi Y. The Curious Case of Neural Text Degeneration. URL: https://arxiv.org/abs/1904.09751?spm=a2ty_o01.29997173.0.0.1f0d5171Q1vJzw&file=1904.09751.ASd

Information about the authors

PUGACHEV Andrey Sergeevich, Postgraduate student, junior researcher at the laboratory of robotic systems, engineer
Institute of Marine Technology Problems, Far Eastern Branch of Russian Academy of Science
Address: Russia, 690091, Vladivostok, Sukhanova str., 5A
Research interests: computing systems, compilers, neural networks, application software
E-mail: pugachevas@marine.febras.ru
Phone: +7(423)2-215-545, ext. 509

BOROVIK Alexey Igorevich, Candidate of Technical Sciences, Leading Researcher, Head of the Laboratory of Robotic Systems
Institute of Marine Technology Problems, Far Eastern Branch of Russian Academy of Science
Address: Russia, 690091, Vladivostok, Sukhanova str., 5A
Research interests: AUV, ROV, robot control systems, robotic software platforms, software development
E-mail: alexey@borovik.me
Phone: +7(423)2-215-545, ext. 510
ORCID: 0000-0002-9696-2751