Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский физико-технический институт
(национальный исследовательский университет)»
Физтех-школа Прикладной Математики и Информатики
Кафедра корпоративных информационных систем

Направление подготовки / специальность: 03.04.01 Прикладные математика и физика
Направленность (профиль) подготовки: Цифровая инженерия, информационные технологии и дискретная математика

МАШИННОЕ ОБУЧЕНИЕ ДЛЯ ФОРМИРОВАНИЯ КОНТЕКСТНОЙ ПОДСКАЗКИ (КОНТЕНТ-АССИСТ) ПРИ РАЗРАБОТКЕ НА ПЛАТФОРМЕ 1С:ПРЕДПРИЯТИЕ

(магистерская диссертация)

Студент:
Каразеев Антон Андреевич

_____
(подпись студента)

Научный руководитель:
Дзюба Максим Владимирович,

_____
(подпись научного руководителя)

Консультант *(при наличии)*:
_____
_____
(подпись консультанта)

Москва 2021

Оглавление

Стр.

Стр.

Аннотация

В магистерской дипломной работе исследуются механизмы построения контекстной подсказки и предлагается алгоритм, повышающий релевантность контекстной подсказки встроенного языка платформы «1С: Предприятие 8» при помощи методов машинного обучения. За основу алгоритма используется модель code2seq и в качестве входных данных рассматривается абстрактное синтаксическое дерево (АСД).

При реализации инструмента для построения контекстной подсказки исследуется предметная область - встроенный язык платформы «1С: Предприятие 8», математическая модель представления кода программы, модель машинного обучения для ранжирования элементов контекстной подсказки.

Описан процесс встраивания полученного алгоритма в качестве плагина для среды разработки 1C Enterprise Development Tools (1C:EDT).

Введение

Построить контекстную подсказку для языков со статической типизацией довольно просто, чего нельзя сказать про языки с динамической типизацией. Во втором случае типы переменных и возвращаемые типы методов определяются не сразу в момент написания, а лишь на этапе выполнения программы и могут меняться в процессе выполнения. То есть синтаксический анализ, компиляция и определение типов данных происходит ”на лету”.

С помощью правильно подготовленных данных, полученных из уже написанных программ, и методов машинного обучения можно повысить качество кода, снизить количество ошибок при разработке программ и облегчить жизнь разработчику.

В магистерской работе рассматриваются алгоритмы, с помощью которых можно ранжировать элементы контекстной подсказки для её формирования, а также проводятся эксперименты с выбранным алгоритмом на текстах конфигураций для платформы «1С: Предприятие 8».

Описывается возможная реализация плагина для среды разработки 1C Enterprise Development Tools (1C:EDT), который содержит в себе построенный алгоритм ранжирования контекстной подсказки.

Актуальность работы. При разработке конфигураций для платформы «1С: Предприятие 8» разработчику приходится оперировать большим количеством различных элементов (переменные, методы и объекты метаданных). Если повысить релевантность выдачи контекстной подсказки, то удастся оптимизировать время разработчика - набор текста конфигурации будет происходить быстрее и с меньшим количеством ошибок.

Объект работы. Повышение релевантности элементов, предсказанных контекстной подсказкой.

Предмет работы. Использование машинного обучения при формировании выдачи контекстной подсказки при разработке на платформе «1С: Предприятие 8».

Цель работы. При написании кода программы разработчику приходится иметь дело с большим количеством различных переменных и методов. Для упрощения и оптимизации его работы существует контекстная подсказка (контент-ассист), которая предлагает наиболее релевантные объекты в данном контексте

программы. Целью этой дипломной работы является разработка контекстной подсказки для платформы «1С: Предприятие 8» с применением машинного обучения.

Задачи работы. Необходимо решить следующие задачи, чтобы достичь поставленной цели:

1. Исследовать имеющиеся решения для построению контекстной подсказки в различных языках программирования;
2. Проанализировать существующие алгоритмы, которые могут быть использованы для формирования релевантной контекстной подсказки в языках с динамической типизацией;
3. Исследовать предметную область и реализовать алгоритм построения контекстной подсказки с применением машинного обучения для предметно-ориентированного языка платформы «1С: Предприятие 8»;
4. Провести эксперимент с разработанным алгоритмом на текстах конфигураций;
5. Описать процесс встраивания полученного алгоритма в качестве плагина для 1C Enterprise Development Tools (1C:EDT).

Научная новизна работы. Применение машинного обучения к задаче повышения релевантности выдачи контекстной подсказки - новый подход в этой области.

Работа состоит из аннотации, введения и пяти глав. Полный объём работы составляет 58 страниц, включая 24 рисунка и 5 таблиц. Список литературы содержит 13 наименований.

Глава 1. Обзор существующих решений

1.1 Существующие решения для построения контекстной подсказки

Рассмотрим наиболее известные:

1. Eclipse Code Recommenders[1]:
   - Проект архивирован, более не поддерживается;
   - Статистические алгоритмы;
2. Eclipse Orion[2]:
   - Заточен под веб-разработку;
   - Разработан с учетом особенностей языка программирования JavaScript;
3. Jedi[3]:
   - Разработан для языка программирования Python;
   - Статический анализатор на основе синтаксического дерева;
   - 120 соавторов, 4.9k звезд на GitHub.

1.1.1 Контент-ассист для языка программирования JavaScript

Цель проекта Eclipse Orion - создать платформу интеграции открытых инструментов внутри браузера, которая была бы полностью ориентирована на разработку для Интернета. Инструменты написаны на JavaScript и запускаются в браузере. Большое внимание уделено веб-интерфейсу для разработки, а не воссозданию традиционного интерфейса IDE во вкладке браузера. Для синтаксического анализа используется парсер Esprisma[4].

[1] https://projects.eclipse.org/projects/technology.recommenders
[2] https://projects.eclipse.org/projects/ecd.orion
[3] https://github.com/davidhalter/jedi
[4] https://esprima.org

Принципы работы

Принцип работы контент-ассиста основан на абстрактном синтаксическом дереве (АСД) [5]. Благодаря этому предложения контекстной подсказки более релевантны. Можно выделить следующие этапы:

1. При вызове контент-ассиста Esprisma анализирует содержимое буфера;
2. Плагин обходит построенное АСД;
3. При обходе дерева АСД записывается целевой тип каждого узла, а также назначения и объявления. Эта информация помогает учитывать доступные свойства для каждого известного типа в любой момент времени при обходе АСД;
4. После обхода достаточной части АСД (нет необходимости в обходе всего дерева, поскольку некоторые его части не будут релевантными для данного вызова контент-ассиста) рассчитываются все доступные варианты на основании целевого типа.

Поддерживаемые функции

1. Распознавание областей функции: соблюдается область видимости; идентификаторы, недоступные в текущей области, не отображаются в окне контент-ассиста;
2. Объектные литералы: предлагаются пары ключ-значение и даже распознаются литералы вложенных объектов;
3. Учитывается простой порядок выполнения, поэтому назначение (переменных) запоминается;
4. Доступны некоторые предопределенные типы (JSON, Number, String, Boolean, …);
5. Конструкторы: функции, начинающиеся с больших букв, считаются конструкторами;
6. Устойчивость к неправильно написанному коду: обычно при вводе имени переменной и последующего знака ”.” пользователь ожидает, что

[5] https://en.wikipedia.org/wiki/Abstract_syntax_tree

контент-ассист предоставит все подходящие варианты, однако большинство парсеров JavaScript перестают работать после первой же ошибки в написании кода.

1.1.2 Контент-ассист для языка программирования Python

Наиболее распространённым и популярным контент-ассистом является Jedi[6] - инструмент статического анализа, который обычно используется в плагинах IDE. Функционал Jedi включает в себя автодополнение, рефакторинг, поиске кода и ссылок.

Для начала рассмотрим последовательность действий, которым следует интерпретатор Python при выполнении программы:

1. Код разбирается на части, которые обычно называют токенами. Эти токены основаны на разных правилах - например, ключевое слово ”if” и числовое значение ”42” являются разными токенами;
2. Список токенов используется для построения АСД, в котором вершины связаны между собой грамматикой языка Python;
3. По АСД интерпретатор создает низкоуровневые инструкции (байт-код);
4. Имея в своём распоряжении низкоуровневые инструкции, интерпретатор может наконец запустить код программы.

В качестве встроенного парсера Jedi использует parso[7]. С его помощью создается синтаксическое дерево, которое Jedi анализирует и пытается понять. Грамматика, которую использует этот синтаксический анализатор, очень похожа на официальные файлы грамматики Python[8].

Определение типа в Jedi основано на трёх предположениях:

1. Код использует как можно меньше сторонних эффектов. Jedi понимает определенные модификации встроенных классов list/tuple/set, но нет никакой гарантии, что Jedi обнаружит все;
2. Используются встроенные элементы:
   - метаклассы;
   - setattr() / __import__();
   - запись в globals(), locals(), object.__dict__;

Алгоритм основан на принципе «ленивый вывод типа» - принцип хорош тем, что он выводит только то, что должно быть выведено; а все операторы и модули, которые не используются, просто игнорируются. Тем не менее, типичной

[6] https://github.com/davidhalter/jedi
[7] https://github.com/davidhalter/parso
[8] https://docs.python.org/3/reference/grammar.html

точкой входа для статического анализа является вызов infer_expr_stmt. В API есть отдельная логика для автодополнения - inference_state[9].

[9] https://jedi.readthedocs.io/en/latest/docs/development.html

1.2 Существующие алгоритмы для построения контекстной подсказки

Можно выделить следующие классы алгоритмов для построения контекстной подсказки:

- Синтаксический анализатор (не учитывает контекст, может предлагать редко используемые элементы);
- Статистический подход - предлагает элементы на основании аналитических данных множества проектов (не учитывает контекст).

Представленные ранее существующие решения, а также классы алгоритмов объединяет следующее:

- Завязаны на конкретный язык и базу знаний о нём;
- Нет возможности переноса опыта на другие языки;
- Не учитывают контекст.

На помощь приходят алгоритмы машинного обучения. Последние достижения в этой области говорят о том, что можно построить модель, которая будет аппроксимировать с заданной точностью функцию отображения контекста и места вызова контекстной подсказки в множество доступных элементов кода программы. А также переупорядочивать их наиболее релевантным образом.

Для достижения этой цели необходимо сформулировать задачу в терминах машинного обучения, собрать тренировочные и тестовые данные, а затем обучить модель. На тренировочных данных модель будет обучаться, извлекая полезную информацию и закономерности из предоставленных данных. А на тестовых данных будет проводиться оценка качества обучения - насколько хорошо модель обобщает полученные знания.

1.2.1 Условные случайные поля

Авторы в «Predicting Program Properties from ”Big Code”»[1] используют условные случайные поля для моделирования различных взаимосвязей между переменными, элементами АСД и типами для предсказания имён и типов переменных (например, для деобфускации приложений Android), но без явного учета потока данных. В этих работах все использования переменных известны заранее детерминированно (поскольку код является полным и остается неизмененным).

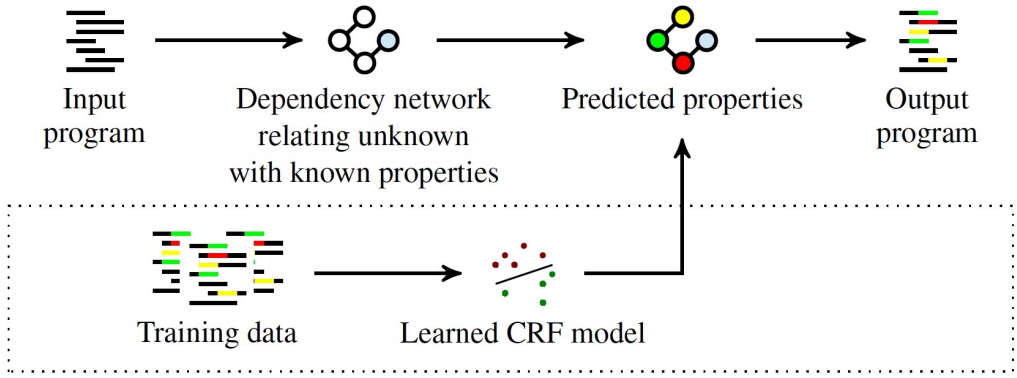


Рисунок 1.1 — Статистическое предсказание параметров программы [1].

Начнём с определения. Условное случайное поле (Conditional random field, CRF) - один из методов статистического моделирования, который часто применяется в распознавании образов и используется для структурированного предсказания.

Пусть $x \in X$ - текст программы из множества программ X. Обозначим за $Labels$ набор всевозможных значений, которые может принимать какое-либо свойство (например, в случае предсказания типа $Labels$ будет состоять из {число, строка, ...}). Тогда для программы x можно определить $\boldsymbol{y}$ - вектор из свойств, соответствующих данной программе ($\boldsymbol{y} \in Y$, где $Y = (Labels)^*$).

Таким образом, можно сформулировать задачу как поиск наиболее вероятного вектора свойств $\boldsymbol{y}$ при помощи метода оценки апостериорного максимума (Maximum a posteriori estimation, MAP)[10]:

$$\boldsymbol{y} = argmax_{\boldsymbol{y}' \in \Omega_x} Pr(\boldsymbol{y}'|x),$$

[10] https://en.wikipedia.org/wiki/Maximum_a_posteriori_estimation

где $\Omega_x \subseteq Y$ - множество возможных значений для вектора свойств $\boldsymbol{y}'$ для элементов программы x.

Опишем CRF как модель для представления условной вероятности $Pr(\boldsymbol{y}|x)$. Без потери общности рассмотрим случай, в котором переменные принимают положительные значения, тогда условная вероятность свойств $\boldsymbol{y}$ при данной программе x может быть выражена как:

$$Pr(\boldsymbol{y}|x) = \frac{1}{Z(x)} \exp(score(\boldsymbol{y}, x)),$$

где $score$ - функция, возвращающая действительное число и которая сообщает о принадлежности свойств $\boldsymbol{y}$ к программе x (чем выше её значение, тем выше вероятность принадлежности), $Z(x) = \sum_{\boldsymbol{y} \in \Omega_x} \exp(score(\boldsymbol{y}, x))$.

Рассматриваются такие $score$ функции, которые могут быть представлены как сумма k функций f_i, связанных с весами w_i:

$$score(\boldsymbol{y}, x) = \sum_{i=1}^{k} w_i f_i(\boldsymbol{y}, x) = \boldsymbol{w}^T \boldsymbol{f}(\boldsymbol{y}, x),$$

где $\boldsymbol{f}$ - вектор функций f_i и $\boldsymbol{w}$ - вектор весов w_i, $f_i : Y \times X \to \mathbb{R}$. Веса $\boldsymbol{w}$ могут быть выучены из исходных данных.

Модель для условной вероятности свойств $\boldsymbol{y}$ при данных наблюдениях x называется условным случайным полем и может быть представлена как:

$$Pr(\boldsymbol{y}|x) = \frac{1}{Z(x)} \exp(\boldsymbol{w}^T \boldsymbol{f}(\boldsymbol{y}, x)).$$

Ограничения

Обобщающая способность. Авторы статей «Predicting Program Properties from ”Big Code”»[1] и «A General Path-Based Representation for Predicting Program Properties»[2] обнаружили, что CRF являются мощным методом для предсказания свойств программ. Однако они ограничиваются моделированием тех комбинаций значений, которые уже были в обучающих данных. Когда в тестовых данных попадается невиданная ранее комбинация, модели такого рода не могут обобщить

и оценить вероятность этой комбинации, даже если каждое отдельное значение присутствовало в тренировочных данных.

Полиномиальная сложность. Использование CRF в такой задаче, которая моделирует вероятность метки с учетом набора контекстов пути фрагмента кода, потребует сохранение параметров для каждой наблюдаемой комбинации из четырех компонентов: терминальный элемент АСД, другой терминальный элемент, путь между ними и целевая метка фрагмента кода. Таким образом, CRF будет иметь пространственную сложность $O(|X|^2 \cdot |P| \cdot |Y|)$, где X - набор переменных из фрагментов кода, P - набор путей, а Y - набор меток.

1.2.2 Графовые нейронные сети

Графовая нейронная сеть (Graph neural network, GNN) - это такая нейронная сет, которая работает со структурой графа напрямую. Основным применением GNN является классификация узлов. Графовые нейронные сети были впервые предложены в 2009 году в работе «The Graph Neural Network Model»[3], которая расширила существующие нейронные сети для обработки данных, представленных в графовых областях.

При работе с естественными языками, обработке и анализе изображений, построении моделей веб-сетей и еще широком спектре прикладных задач, бывает удобно представлять данные в виде графов:

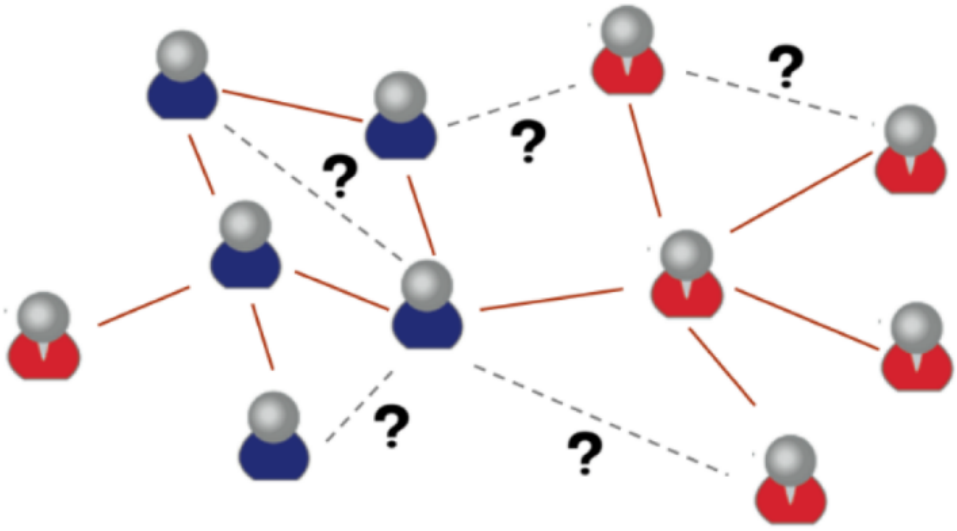


Рисунок 1.2 — Пример задачи на графах: социальные сети.

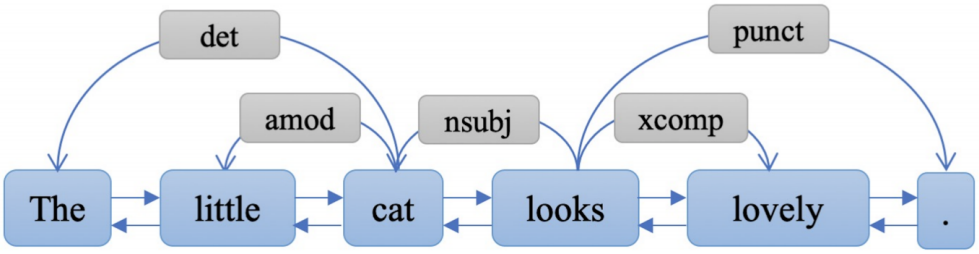


Рисунок 1.3 — Пример задачи на графах: анализ текста.

Однако для традиционных методов машинного обучения необходимо предварительно преобразовывать графово структурированные данные в другие струк-

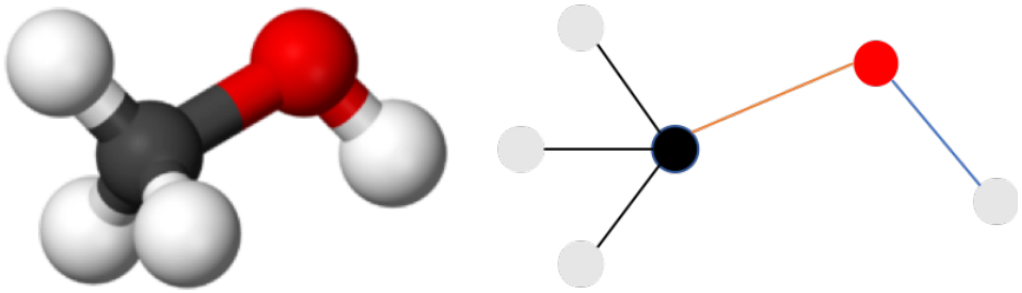


Рисунок 1.4 — Пример задачи на графах: расчёт свойств молекул.

туры данных, к примеру числовой вектор. Такой подход может привести к потере части информации, заключающейся во взаиморасположении узлов сети.

Использование GNN позволяет работать с данными графов, без предварительной обработки. Такой подход позволяет сохранить топологические отношения между узлами графа.

В графе каждый узел определяется его признаками и связанными узлами. Целью GNN является изучение состояния встраивания $\boldsymbol{h}_v \in \mathbb{R}^s$, которое содержит информацию об окрестностях для каждого узла. Состояние встраивания $\boldsymbol{h_v}$ - это s-мерный вектор из вершины v, который может быть использован для получения выхода $\boldsymbol{o_v}$ (метки узла).

Пусть f - это параметрическая функция, называемая локальной переходной функцией, которая является общей для всех узлов и обновляет состояние узла в соответствии с входной окрестностью. Также пусть g - локальная выходная функция, которая описывает, как выход был производен. Тогда $\boldsymbol{h_v}$ и $\boldsymbol{o_v}$ определяются следующим образом:

$$\boldsymbol{h_v} = f(\boldsymbol{x_v}, \boldsymbol{x}_{co[v]}, \boldsymbol{h}_{ne[v]}, \boldsymbol{x}_{ne[v]}),$$
$$\boldsymbol{o_v} = g(\boldsymbol{h_v}, X_v),$$

где $\boldsymbol{x_v}, \boldsymbol{x}_{co[v]}, \boldsymbol{h}_{ne[v]}, \boldsymbol{x}_{ne[v]}$ являются признаками v, признаками его ребер, состояний, и признаками узлов в окрестностях v соответственно.

Пусть H, O, X и X_N - это векторы, построенные путем укладки всех состояний, всех выходных данных, всех признаков и всех признаков узлов соответственно. Тогда мы можем представить уравнения в более компактной форме:

$$H = F(H, X),$$
$$O = G(H, X_N),$$

где F - это глобальная функция перехода, а G - глобальная выходная функция, которая является сложенной версией f и g для всех узлов в графе соответственно. Значение H является фиксированным и однозначно определяется в предположением о том, что F - это карта сжатия.

С учетом теоремы Банаха о неподвижной точке GNN использует следующую классическую итерационную схему для вычисления состояния: $H^{t+1} = F(H_t,X)$, где H_t обозначает t-ую итерацию H. Динамическая система из этого уравнения сходится экспоненциально быстро к решению уравнения $H = F(H, X)$ для любого начального значения $H(0)$. Стоит отметить, что вычисления, описанные в f и g, могут быть интерпретированы как нейронные сети с прямой связью.

Следующие задачей становится поиск параметров f и g. С помощью целевой информации (t_v для конкретного узла) для контроля, функция потерь может быть вычислена следующим образом:

$$loss = \sum_{i=1}^{p} (\boldsymbol{t}_i - \boldsymbol{o}_i),$$

где p - это число контролируемых узлов. Алгоритм обучения основан на стратегии градиентного спуска и состоит из следующих шагов:

- Состояния h_v^t итеративно обновляются $\boldsymbol{h_v} = f(\boldsymbol{x_v}, \boldsymbol{x}_{co[v]}, \boldsymbol{h}_{ne[v]}, \boldsymbol{x}_{ne[v]})$ до времени T. Они стремятся к фиксированному значению $H(T) \approx H$;
- Градиент весов W вычисляется из функции потерь;
- Веса W обновляются в соответствии с градиентом, вычисленным на последнем шаге.

Применение

Авторы «Learning to Represent Programs with Graphs»[4] предлагают использовать графы для представления синтаксической и семантической структур кода и использовать графовые методы машинного обучения (включая графовые нейронные сети) для решения ряда задач, связанных с предсказаниями на основе текстов программ.

Более того, авторы предлагают применять модифицированную версию графовой нейронной сети, Gated graph neural network, которая была представлена в «Gated Graph Sequence Neural Networks»[5].

Описать предлагаемое решение можно следующим образом. Граф $G = (V, \boldsymbol{E}, \boldsymbol{X})$ состоит из множества вершин V, признаков узлов $\boldsymbol{X}$ и набора направленных рёбер $E = (E_1, \ldots, E_K)$, где K - число типов рёбер. Каждой вершине $v \in V$ ставится в соответствие действительный вектор $\boldsymbol{x}^{(v)} \in \mathbb{R}^D$, представляющий признаки узла (например, эмбеддинг метки узла).

Вектор состояния $\boldsymbol{h}^{(v)}$ ставится в соответствие каждой вершине v на основании метки вершины $\boldsymbol{x}^{(v)}$. Для распространения информации по графу отправляются "сообщения"типа k от вершины v к соседним вершинам, где "сообщение"вычисляется на основании текущего вектора состояния как $\boldsymbol{m}_k^{(v)} = f_k(\boldsymbol{h}^{(v)})$. В качестве f_k может выступать произвольная функция (например, линейная). Новое состояние для вершины v вычисляется согласно формуле $\widetilde{\boldsymbol{m}}^{(v)} = g(\{\boldsymbol{m}_k^{(u)} | \text{существует ребро типа k из u в v}\})$, где g - функция агрегации (авторы в качестве g использовали поэлементное сложение). При данных $\widetilde{\boldsymbol{m}}^{(v)}$ и $\boldsymbol{h}^{(v)}$ можно вычислить состояние на следующей итерации как $\boldsymbol{h}'^{(v)} = GRU(\widetilde{\boldsymbol{m}}^{(v)}, \boldsymbol{h}^{(v)})$, где GRU - управляемый рекуррентный блок, представленный в статье «On the Properties of Neural Machine Translation: Encoder–Decoder Approaches»[6].

Динамика, определяемая уравнениями сверху, повторяется определённое количество итераций. А затем векторы состояния вершин из последней итерации используются в качестве представления каждой вершины.

Ограничения

Несмотря на то, что экспериментальные результаты показали, что GNN является мощной архитектурой для моделирования структурных данных, существует еще несколько ограничений для обычного GNN.

Во-первых, итеративно обновлять скрытые состояния узлов для фиксированной точки неэффективно. Если ослабить предположение о неподвижной точке, мы можем спроектировать многослойную GNN, чтобы получить стабильное представление узла и его окрестности.

Во-вторых, GNN использует одни и те же параметры в итерации, в то время как большинство популярных нейронных сетей используют разные параметры в разных слоях, которые служат в качестве метода извлечения иерархических признаков.

В-третьих, есть также некоторые информативные признаки на краях, которые не могут быть эффективно смоделированы обычной GNN. Например, ребра в графе знаний имеют тип отношений, и распространение сообщения через разные ребра должно быть различным в зависимости от их типов. Кроме того, поиск скрытых состояния ребер, также является важной проблемой.

Наконец, нецелесообразно использовать фиксированные точки, если мы сосредоточимся на представлении узлов вместо графиков, поскольку распределение представления в фиксированной точке будет гораздо более гладким по значению и менее информативным для различения каждого узла.

1.2.3 code2seq

Возможность генерировать последовательности естественного языка из фрагментов исходного кода имеет множество приложений, таких как обобщение кода, генерация документации и поиск. Модели Sequence-to-sequence (seq2seq[11]), заимствованные из нейронного машинного перевода (NMT[12]), достигли достаточных показателей в этих задачах, рассматривая тексты программ как последовательности токенов. Авторы «code2seq: Generating Sequences from Structured Representations of Code»[7] представляют code2seq: альтернативный подход, который использует синтаксическую структуру языков программирования для лучшего кодирования текстов программ. Модель представляет фрагменты кода в виде набора путей по его абстрактному синтаксическому дереву (АСД) и использует механизм внимания для выбора соответствующих путей при декодировании. Представленная модель значительно превосходит предыдущие модели, которые были специально разработаны для языков программирования, а также современные модели NMT.

АСД однозначно определяет фрагмент исходного кода. Листьями такого дерева выступают переменные, идентификаторы класса и прочие определённые пользователем элементы - такие вершины также называются терминальными. Внутренние вершины дерева соответствуют структурам языка программирования: циклы, выражения и объявления переменной.

Например, на рисунке 1.5c показан частичный АСД для фрагмента кода на рисунке 1.5a. Имена (например, num) и типы (например, int) представлены как значения терминальных вершин; синтаксические структуры, такие как объявление переменной (VarDec) и цикл do-while (DoStmt), представлены как внутренние вершины.

Рассматриваются все попарные пути между терминалами и представляются в виде последовательностей терминальных и внутренних вершин. Затем эти пути вместе со значениями их вершин начала и конца используются для представления самого фрагмента кода. Например, рассмотрим два метода Java на рисунке 1.5. Оба этих метода подсчитывают вхождения символа в строку. Они имеют точно такую же функциональность, хотя и различную реализацию, и, следовательно,

[11] https://en.wikipedia.org/wiki/Seq2seq

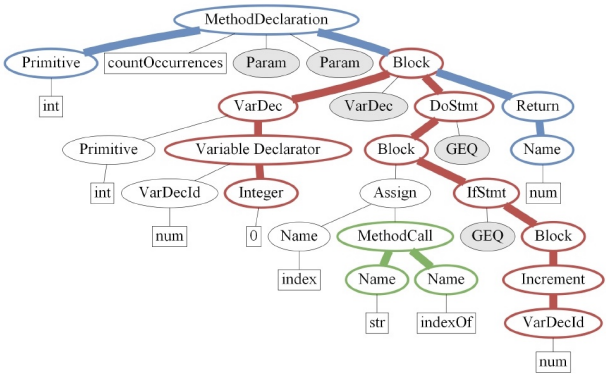
[12] https://en.wikipedia.org/wiki/Neural_machine_translation

```java
int countOccurrences(String str, char ch) {
    int num = 0;
    int index = -1;
    do {
        index = str.indexOf(ch, index + 1);
        if (index >= 0) {
            num++;
        }
    } while (index >= 0);
    return num;
}
```

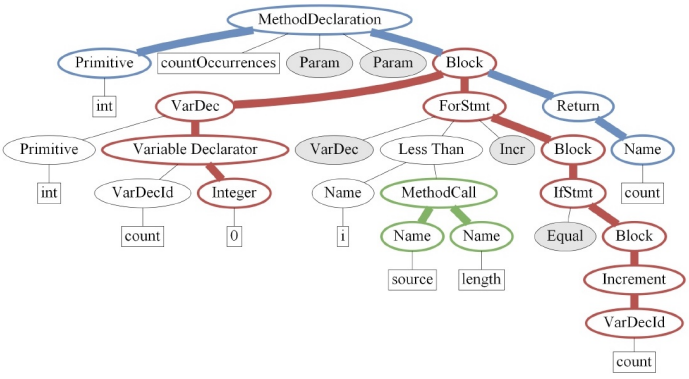
(a)

```java
int countOccurrences(String source, char value) {
    int count = 0;
    for (int i = 0; i < source.length(); i++) {
        if (source.charAt(i) == value) {
            count++;
        }
    }
    return count;
}
```

(b)



(c)



(d)

Рисунок 1.5 — Пример двух методов на языке программирования Java, которые выполняют одну и ту же функцию. Несмотря на то, что эти методы имеют различные последовательные представления (основанные на токенах), если мы рассмотрим синтаксическое представление, то обнаружим повторяющиеся пути, которые могут отличаться только в одном узле (узел ForStmt вместо узла Do-while), будут выявлены [7].

выглядят немного иначе. Если эти фрагменты рассматривать как последовательности токенов, то схожие закономерности, которые могут говорить об одинаковом функционале, могут быть пропущены. Однако структурное наблюдение может выявить синтаксические пути, которые являются схожими для обоих функций и отличаются только узлами оператора Do-while и оператора For. Этот пример показывает эффективность синтаксического представления фрагментов кода.

Формально это можно выразить следующим образом. За C обозначим фрагмент кода. На этапе обучения случайным образом выбираются k пар терминальных вершин, которые семплируются из АСД, построенного по фрагменту C. Пара терминалов $(v_1^i, v_{l_i}^i)$ также подразумевает цепочку вершин (путь) между ними: $v_1^i v_2^i \dots v_{l_i}^i$. Наконец, исходный фрагмент кода представляется в виде k случайных АСД путей: $\{(v_1^1 v_2^1 \dots v_{l_1}^1), \dots, (v_1^k v_2^k \dots v_{l_k}^k)\}$, где l_j - длина пути j.

Архитектура модели

Модель code2seq состоит из стандартной для NMT архитектуры encoder-decoder - с той существенной разницей, что encoder считывает входные данные не в виде последовательности токенов, а создает векторное представление для каждого пути из АСД отдельно. Затем decoder проходится по закодированным путям АСД и генерирует целевую последовательность. Архитектура проиллюстрирована на рисунке 1.6.

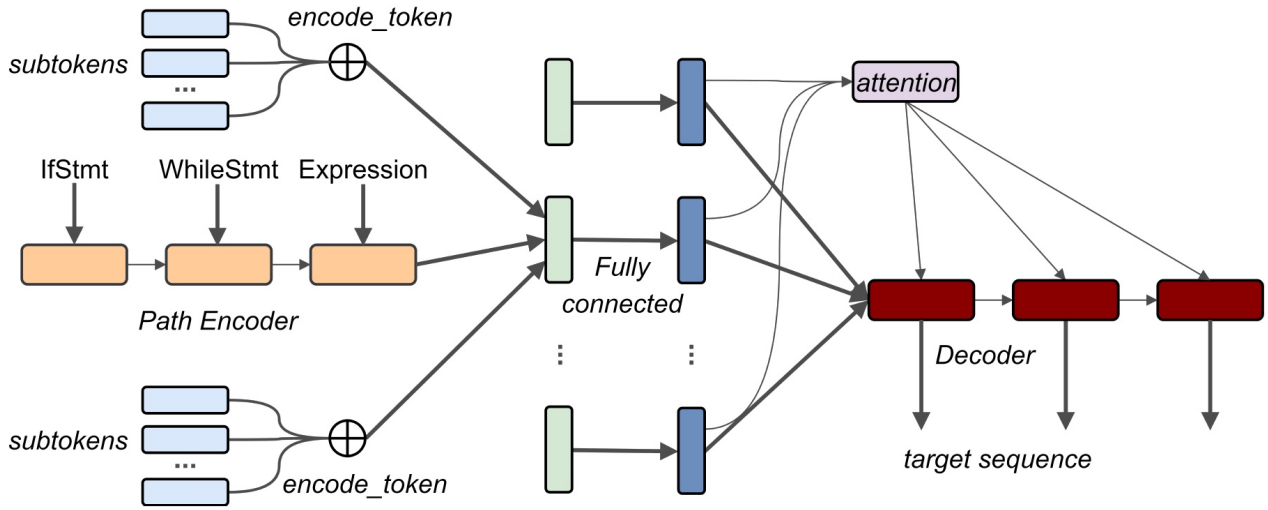


Рисунок 1.6 — Архитектура модели code2seq, которая кодирует входные цепочки АСД в виде векторов и затем предсказывает целевую последовательность [7].

Подход encoder-decoder

Современные модели NMT в значительной степени основаны на архитектуре encoder-decoder [8—11], где encoder отображает входную последовательность токенов $\boldsymbol{x} = (x_1, \ldots, x_n)$ в последовательность непрерывного представления $\boldsymbol{z} = (z_1, \ldots, z_n)$. По заданному $\boldsymbol{z}$ decoder затем генерирует последовательность выходных токенов $\boldsymbol{y} = (y_1, \ldots, y_m)$ по одному токену за раз, моделируя таким образом условную вероятность: $p(y_1, \ldots, y_m | x_1, \ldots, x_n)$.

На каждом этапе декодирования вероятность следующего целевого токена зависит от ранее сгенерированного токена и, следовательно, может быть разложена на множители как:

$$p(y_1, \ldots, y_m | x_1, \ldots, x_n) = \prod_{j=1}^{m} p(y_j | y_{<j}, z_1, \ldots, z_n).$$

В моделях, основанных на внимании, на каждом шаге t в фазе декодирования контекстный вектор c_t вычисляется в зависимости от $\boldsymbol{z}$ с использованием внутреннего состояния decoder h_t, обычно вычисляемого LSTM[13]:

$$\boldsymbol{\alpha}^t = softmax(h_t W_a \boldsymbol{z}),$$

$$c_t = \sum_{i}^{n} \boldsymbol{\alpha}_i^t z_i.$$

Контекстный вектор c_t и состояние декодирования h_t затем объединяются для предсказания текущего целевого токена y_t. Стандартный подход [10] заключается в том, чтобы пропустить c_t и h_t через многослойный перцептрон (MLP[14]), а затем предсказать вероятность следующего токена с помощью функции softmax[15]:

$$p(y_t | y_{<t}, z_1, \ldots, z_n) = softmax(W_s tanh(W_c[c_t; h_t])).$$

АСД encoder

Цель состоит в том, чтобы создать векторное представление z_i для каждого пути $x_i = v_1^i v_2^i \ldots v_{l_i}^i$, учитывая набор путей из АСД $\{x_1, \ldots, x_k\}$. Каждый путь представляется отдельно, используя двунаправленный LSTM для кодирования пути и токенов в виде эмбеддингов.

Представление пути. Каждый путь АСД состоит из узлов и их дочерних вершин из ограниченного словаря. Каждый узел представляется с помощью выученной матрицы эмбеддингов E^{nodes}, а затем кодируется вся последовательность, используя состояния из последней итерации двунаправленного LSTM:

[13] https://en.wikipedia.org/wiki/Long_short-term_memory
[14] https://en.wikipedia.org/wiki/Multilayer_perceptron
[15] https://en.wikipedia.org/wiki/Softmax_function

$$h_1, \ldots, h_l = LSTM(E^{nodes}_{v_1}, \ldots, E^{nodes}_{v_l}),$$

$$encode\_path(v_1, \ldots, v_l) = [h^{\rightarrow}_l; h^{\leftarrow}_1].$$

Представление токенов. Первый и последний узлы пути АСД - это терминальные вершины, значения которых являются токенами в коде (переменные, классы, методы, …). Согласно «Suggesting Accurate Method and Class Names»[12] многословные токены разделяются на составляющие токены; например, токен со значением ArrayList будет разложен на Array и List. Это несколько аналогично кодированию byte-pair в NMT [13]. Для представления каждого подтокена используется матрица эмбеддингов $E^{subtokens}$, а затем полученные векторы суммируются. Таким образом будет построено представление для любого составного токена:

$$encode\_token(w) = \sum_{s \in split(w)} E^{subtokens}_s.$$

Комбинированное представление. Представление пути $x = v_1 \ldots v_l$ получается путём объединения векторного представления пути и эмбеддингов начального и конечного токенов:

$$z = tanh(W_{in}[encode\_path(v_1 \ldots v_l); encode\_token(value(v_l))]),$$

где $values$ - это отображение терминальной вершины к её соответствующему значению, W_{in} - матрица размерности $(2d_{path} + 2d_{token}) \times d_{hidden}$.

Начальное состояние decoder. Представления k путей усредняются, чтобы в итоге получить начальное состояние для decoder:

$$h_0 = \frac{1}{k} \sum_{i=1}^{k} z_i.$$

В отличие от типичных моделей encoder-decoder порядок входных путей не учитывается. Каждый путь кодируется отдельно и объединённые представления затем усредняются, чтобы инициализировать внутреннее состояние decoder. Таким образом фрагмент кода программы представляется как набор случайных путей из АСД.

Механизм внимания. Наконец, decoder генерирует выходную последовательность, посещая все представления $z_1, \ldots, z_k$, аналогично тому, как модели

seq2seq проходятся по исходным последовательностям. Механизм внимания используется, чтобы динамически выбирать распределения над этими k представлениями во время декодирования, точно так же, как модель NMT проходилась бы по закодированным исходным последовательностям.

1.3 Выводы из обзора существующих решений и алгоритмов

Проанализировав существующие решения для формирования контекстной подсказки и релевантные алгоритмы, было принято решение взять за основу модель нейронного машинного обучения code2seq, с помощью которой можно строить векторные представления фрагментов текстов программ и делать предсказания интересующих свойств. Среди прочих преимуществ выбранной модели:

- Не требуется ручной подбор признаков;
- Применимость для различных языков программирования;
- Использование для решения других задач предсказания.

Для обучения модели был составлен датасет, который представляет из себя набор контекстных цепочек между терминальными вершинами АСД:

$$(from\_token, path, to\_token),$$

где $from\_token$ - начальный токен цепочки (операнд), $path$ - последовательность внутренних вершин (операторы) и to_token - конечный токен цепочки (операнд).

Глава 2. Разработка модели машинного обучения для формирования контекстной подсказки

2.1 Формализация задачи

Реализовать инструмент, который будет предлагать N наиболее вероятных ключевых слов при наборе кода программы. Например, рассмотрим фрагмент программы на рисунке 2.1:

```
Запрос = Новый Запрос;
Запрос.<|>
```

Под знаком $<|>$ понимается место вызова контекстной подсказки.

В текущей версии реализации контекстной подсказки пользователю предлагается список, в котором в алфавитном порядке представлены сначала свойства, а потом все возможные методы. Задача заключается в том, чтобы составлять этот список из наиболее подходящих элементов в любой момент времени.

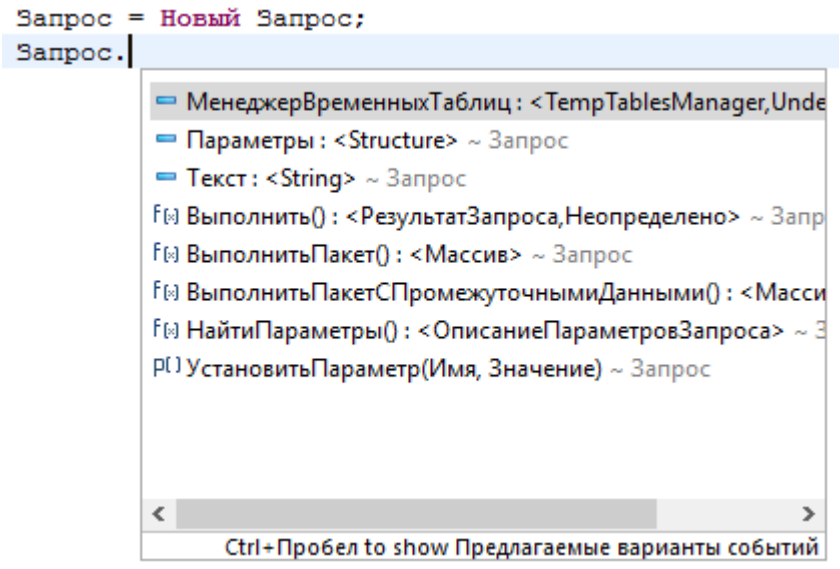


Рисунок 2.1 — Пример выдачи контекстной подсказки.

Можно заметить, что свойство `Текст` находится на третьем месте - хотя именно оно и нужно пользователю с большей вероятностью, а не `Параметры`, которое не имеет смысла без заданного текста. Методы `Выполнить` и `НайтиПараметры` также не имеют смысла без переданного текста в `запрос`.

Таблица 1 — Пример выдачи контекстной подсказки (слева) и желаемая последовательность элементов (справа).

1. МенеджерВременныхТаблиц	1. **Текст**
2. Параметры	2. МенеджерВременныхТаблиц
3. **Текст**	3. УстановитьПараметр()
…	…
8. УстановитьПараметр()	8. Параметры

В таблице 1 приведено сравнение текущей выдачи контекстной подсказки с желаемой, достижение которой и есть основная задача данной работы.

Предлагается решить эту проблему с применением машинного обучения, чтобы на первых местах стояли действительно подходящие варианты.

2.2 Архитектура решения

Предлагаемое решение состоит из предобработки данных, приведения их к требуемому формату, обучения на них модели и затем использования обученной модели на кодах конфигураций платформы «1С: Предприятие 8».

За основу была взята модель нейронного машинного обучения code2seq, представленная в «code2seq: Generating Sequences from Structured Representations of Code»[7].

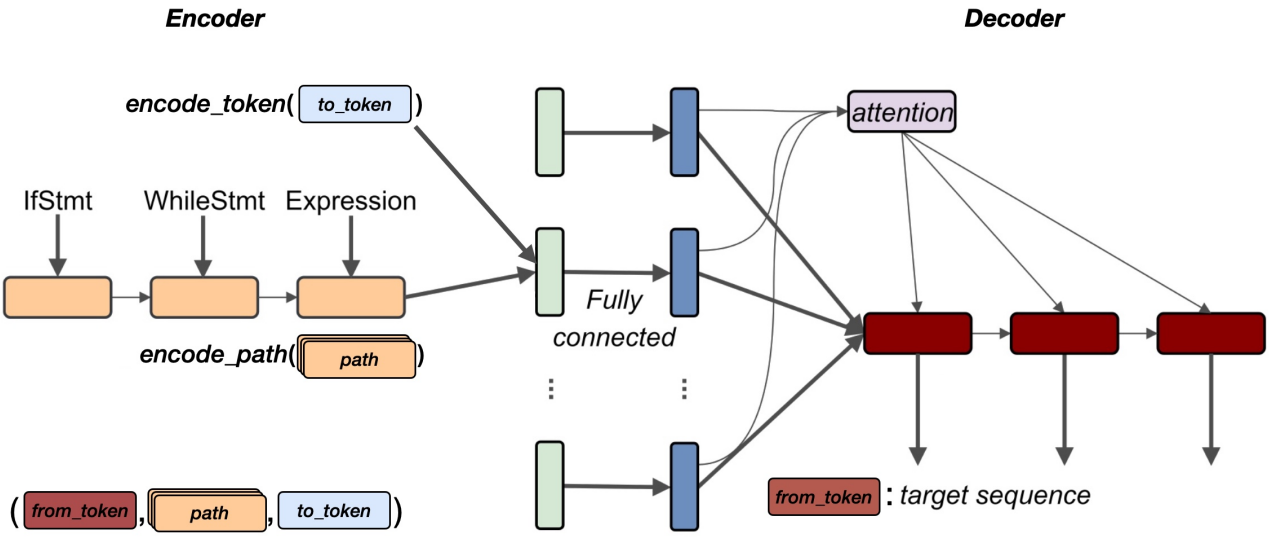


Рисунок 2.2 — Архитектура итоговой модели, которая за основу использует модель code2seq [7]. Модель кодирует входные цепочки АСД, состоящие из path и to_token, в виде векторов и затем предсказывает целевую последовательность - в нашем случае это from_token.

$$h_2, \ldots, h_l = LSTM(E^{nodes}_{v_2}, \ldots, E^{nodes}_{v_l})$$

$$encode\_path(v_2, \ldots, v_l) = [h_l^{\rightarrow}; h_2^{\leftarrow}]$$

$$encode\_token(w) = \sum_{s \in split(w)} E^{subtokens}_{s}$$

$$z = tanh(W_{in}[encode\_path(v_2 \ldots v_l); encode\_token(value(v_l))])$$

2.3 Данные

2.3.1 Источник

В качестве источника данных были использованы тексты конфигураций для платформы «1С: Предприятие 8».

2.3.2 Подготовка

При помощи скриптов был сформирован набор из 500 файлов на встроенном языке платформы «1С: Предприятие 8». Затем каждый файл был обработан и получены АСД.

2.3.3 Использование

Датасет для обучения состоит из набора контекстных цепочек между листьями АСД:

$$(from\_token, path, to\_token),$$

где from_token - начало цепочки (операнд), path - последовательность внутренних вершин (операторы), to_token - конец цепочки (операнд).

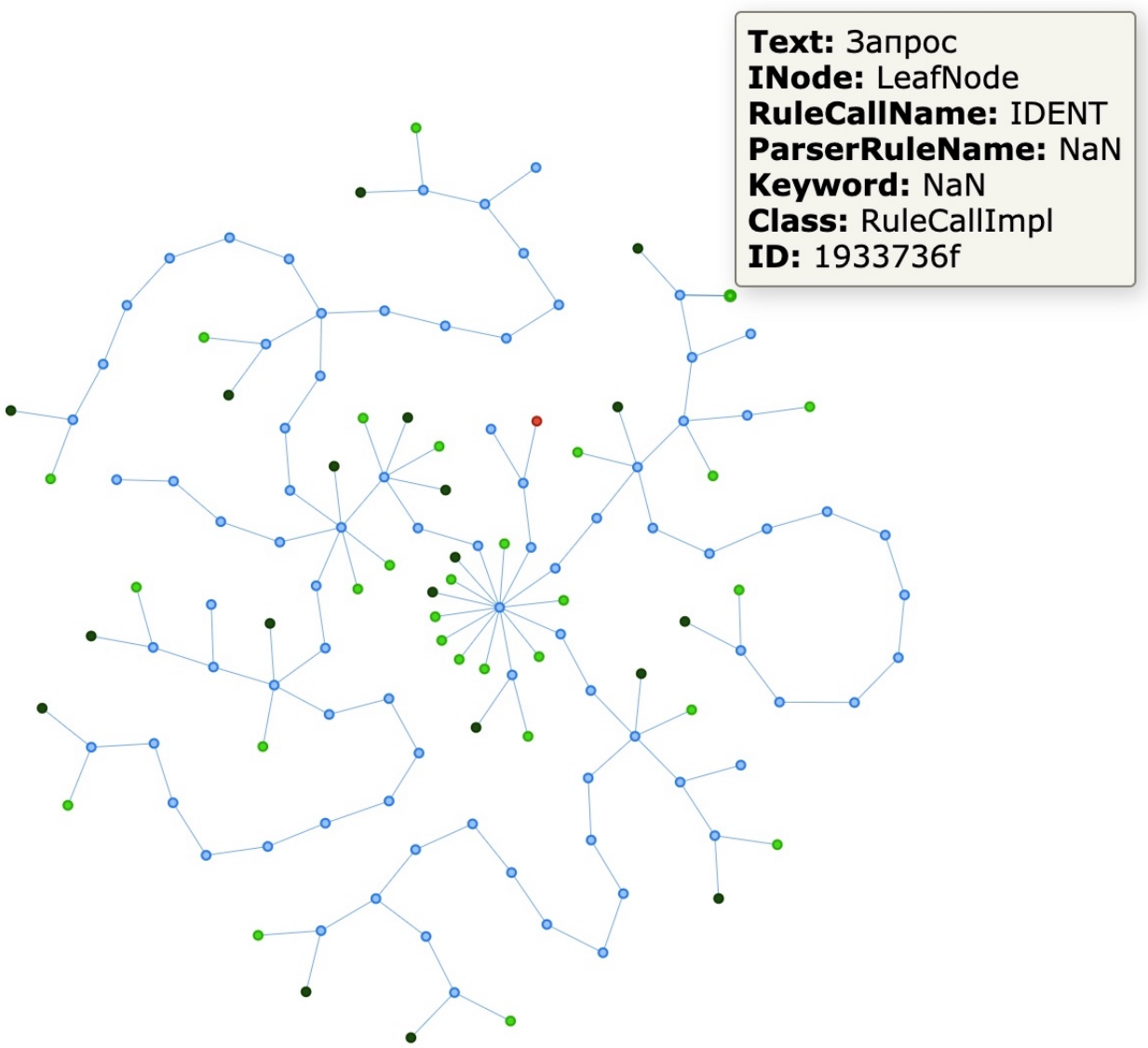


Рисунок 2.3 — АСД до обработки.

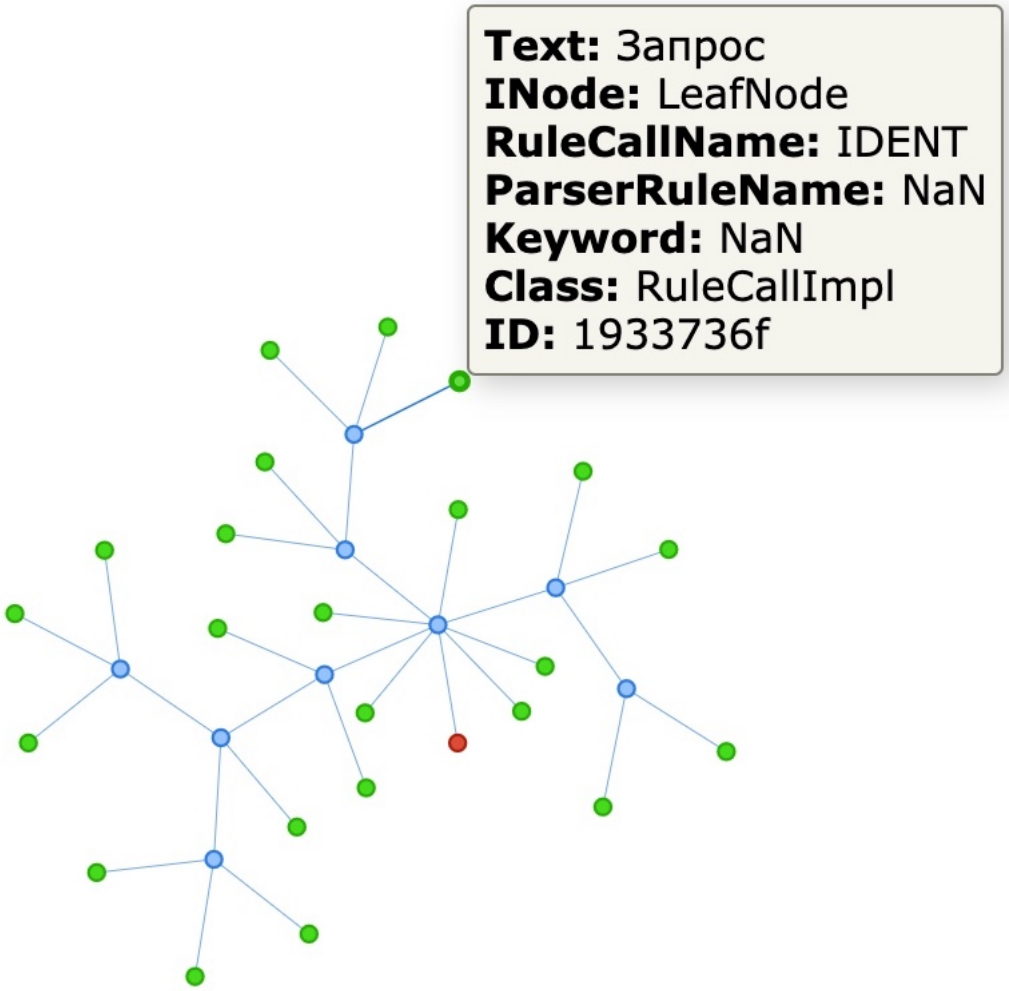


Рисунок 2.4 — АСД после обработки.

Глава 3. Применение алгоритма к формированию контекстной подсказки при разработке на платформе «1С: Предприятие 8»

3.1 Встроенный язык платформы

Начнём рассмотрение встроенного языка с типа модулей под разные задачи:

- Модуль управляемого приложения - автоматически выполняется в момент загрузки конфигурации;
- Общий модуль - для описания общих алгоритмов для использования в других модулях;
- Модуль прикладного объекта - содержит свойства и методы для работы с объектом внутри встроенного языка;
- Модуль менеджеров объекта - общие методы для объектов одного типа;
- Модуль формы - методы и свойства для обработки событий от элементов формы.

Встроенный язык имеет набор стандартных методов и свойств для работы с данными (глобальный контекст) - язык расширяется согласно объектам, которые созданы в конфигурации. Таким образом, связь между программными модулями является неявной - не указывается через специальные конструкции (include, import). Доступность одного модуля из другого определяется лишь свойством каждого модуля без подразделения на какие-либо другие элементы.

Возможность создания своих типов отсутствует, но можно расширять имеющиеся.

3.1.1 Структура модуля встроенного языка

Каждый программный элемент разрабатываемой конфигурации имеет следующую структуру:

1. Блок переменных модуля;
2. Блок методов модуля;
    2.1. Блок переменных метода;
    2.2. Основной блок метода;
3. Блок инициализации модуля.

Среда разработки предоставляет возможность удобно работать с программными модулями. Для предоставления такой возможности используется Eclipse Text Modeling Framework (Xtext)[1]. Xtext предоставляет полный набор возможностей по созданию собственных Domain-specific languages (DSLs):

– Генератор для парсера DSL;
– Доступ к полученному от парсера синтаксическому дереву разбора;
– Интеграция с Eclipse modeling framework (EMF)[2].

Жизненный цикл обработки модуля встроенного языка. Зависит от того, какую операцию пользователь выполняет в данный момент: построение всего проекта в IDE и редактирование одного или нескольких модулей через редактор.

Рассмотрим подробнее редактирование модулей встроенного языка через редактор. При открытии редактора происходит парсинг текста модуля встроенного языка, в ходе которого мы получаем модель данных модуля.

Модель данных хранит в себе не только информацию, полученную при парсинге текста, переменные, методы и так далее, но и вычисленные - владелец модуля, тип модуля и так далее. Основное назначение построения модели данных при открытии модуля:

– Валидация модели данных;
– Сбор информации о используемых объектах конфигурации;
– Контекстная подсказка при наборе кода в модуле.

По умолчанию Xtext рассчитывает допустимые грамматические правила для формирования контекстной подсказки от начала модуля до места вызова подсказки - на больших модулях это очень долго. Поэтому был сокращен объем

[1] https://www.eclipse.org/Xtext/
[2] https://www.eclipse.org/modeling/emf/

анализируемого кода при вызове контекстной подсказки - грамматические правила для контекстной подсказки рассчитываются не с начала модуля, а от начала метода, в котором подсказка была вызвана. Такой подход существенно сократил время расчета допустимых грамматических элементов.

3.1.2 Контекстная подсказка

Контент-ассист это инструмент, созданный для помощи разработчику, содержащий описание всех программных объектов, которые использует система, их методов, свойств, событий и прочего[3]. Синтаксическая помощь автоматически показывается в контекстной подсказке при редактировании модулей конфигурации.

Контекстная подсказка - это инструмент, который помогает разработчику писать и редактировать текст программы. С его помощью возможно ускорить ввод текста и избежать ошибок и опечаток.

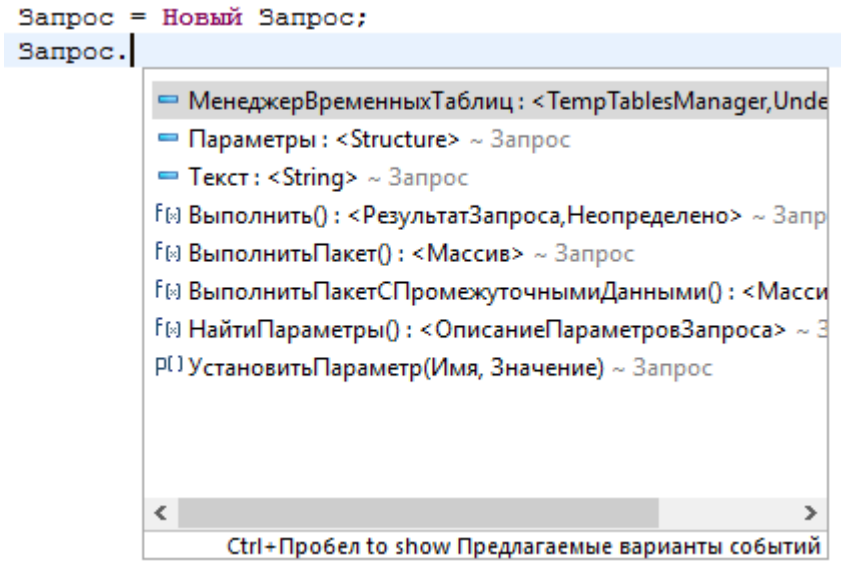


Рисунок 3.1 — Пример выдачи контекстной подсказки для фрагмента кода.

Контекстная подсказка показывает только те свойства, методы и объекты, которые доступны в контексте, который редактируется в текущий момент. В список контекстной подсказки могут быть включены шаблоны текстов и ключевые слова.

Текстовый редактор платформы «1С: Предприятие 8» предоставляет средство контекстного ввода выражений с использованием системных объектов, их свойств, методов, процедур и функций, наименований объектов, определенных в конфигурации, а также переменных, процедур и функций, определенных в общих модулях, модулях прикладных объектов и модулях форм. В список включаются предопределенные элементы справочников, планов счетов, планов видов характеристик и планов видов расчетов.

[3] 1С:Предприятие 8.3. Практическое пособие разработчика, с. 176

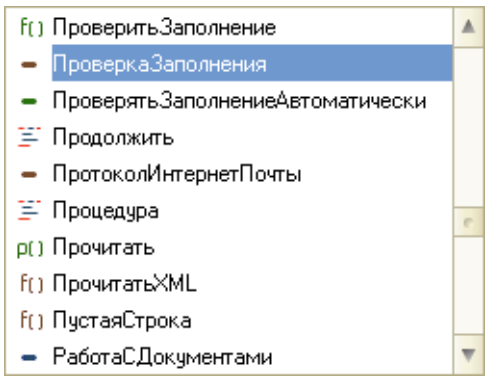


Рисунок 3.2 — Пример выдачи контекстной подсказки, на котором представлены несколько видов пиктограмм, стоящих в начале каждой строчки.

Список контекстной подсказки включает имена системных перечислений, предопределенных элементов, свойств и методов программных объектов, наименования объектов, определенных в конфигурации, и пр. В начале каждой строки списка присутствуют пиктограммы, указывающие на вид объекта: например, черная черта – это свойства глобального контекста, красная черта – локальные переменные модуля, зеленые символы Р() – процедуры прикладных объектов и так далее. Некоторые из перечисленных типов пиктограмм представлены на рисунке 3.2.

3.1.3 Описание работы алгоритма в качестве плагина

Описание работы алгоритма может быть представлено в виде следующих шагов:

1. Плагин встраивается в процесс build проекта через точку расширения Eclipse и обрабатывает каждый модуль встроенного языка;
2. Собирает информацию о контекстах - деревья разбора;
3. Происходит модернизация деревьев и на выходе имеем АСД, по которым уже строятся контекстные цепочки;
4. Цепочки подаются на вход модели, которая предсказывает релевантность элементов контекстной подсказки.

3.2 Обучение модели на подготовленных данных

В данном разделе будет описан процесс обучения модифицированной модели code2seq, архитектура которой представлена на рисунке 2.2, на подготовленных данных.

В таблицах 2 и 3 представлены две конфигурации для encoder и decoder соответственно. Как будет продемонстрировано далее, модель с конфигурацией A удалось лучше натренировать - что подтверждают показатели метрик на тестовой выборке.

Таблица 2 — Гиперпараметры для encoder.

Параметр	Конфигурация A	Конфигурация B
embedding_size	32	10
rnn_size	32	10
use_bi_rnn	true	true
embedding_dropout	0.25	0.25
rnn_num_layers	0.1	0.1
rnn_dropout	0.5	0.5

Таблица 3 — Гиперпараметры для decoder.

Параметр	Конфигурация A	Конфигурация B
decoder_size	32	20
embedding_size	32	10
num_decoder_layers	1	1
rnn_dropout	0.5	0.5
teacher_forcing	1	1
beam_width	0	0

Можно заметить, что скорость выхода на плато функции потерь в процессе обучения у модели с конфигурацией B более высокая - об этом говорит более пологий график на рисунках 3.3 и 3.5.

Параметр max_length, который указывался при составлении датасета, нужен для того, что ограничивать контекстные цепочки по длине сверху: в данной работе были использованы значения max_length=9 и max_length=5.

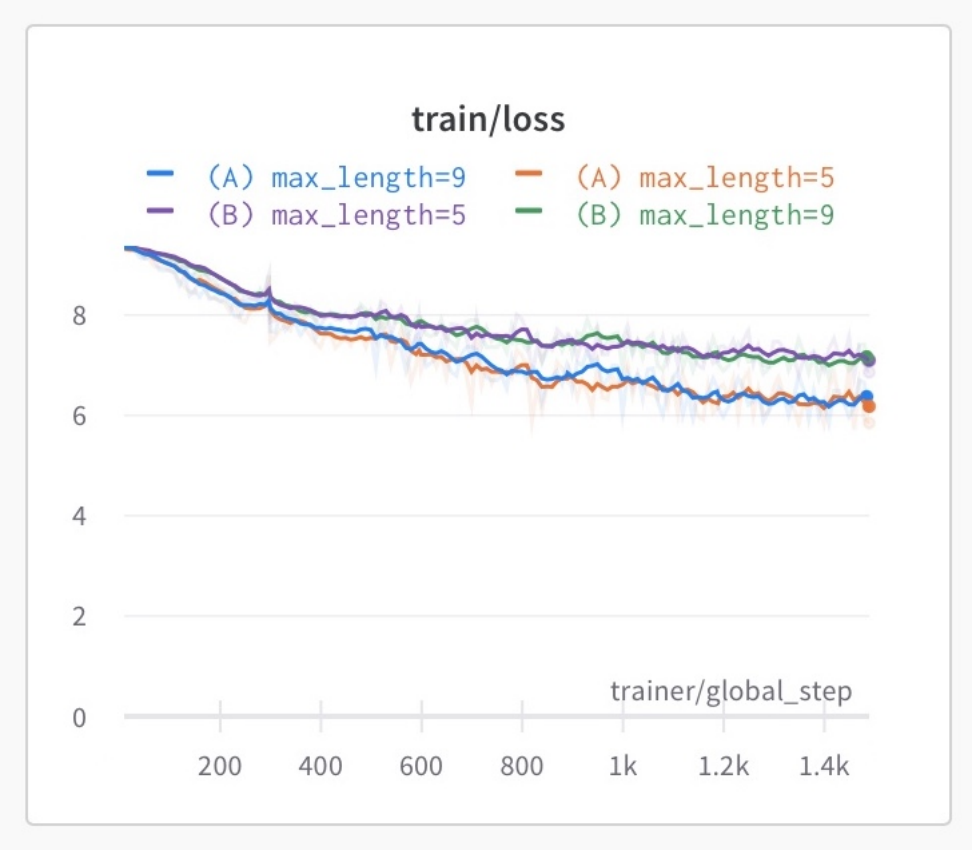


Рисунок 3.3 — Ошибка модели на тренировочной выборке.

Рисунок 3.4 — Значение метрики precision на тренировочной выборке.

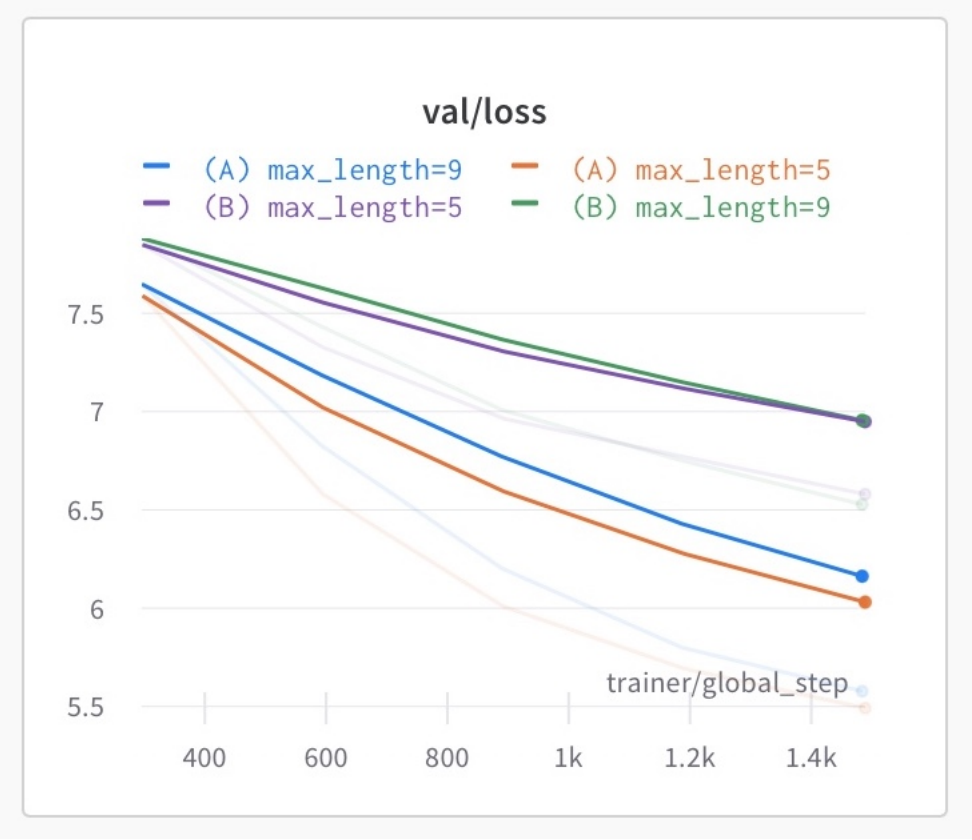


Рисунок 3.5 — Ошибка модели на валидационной выборке.

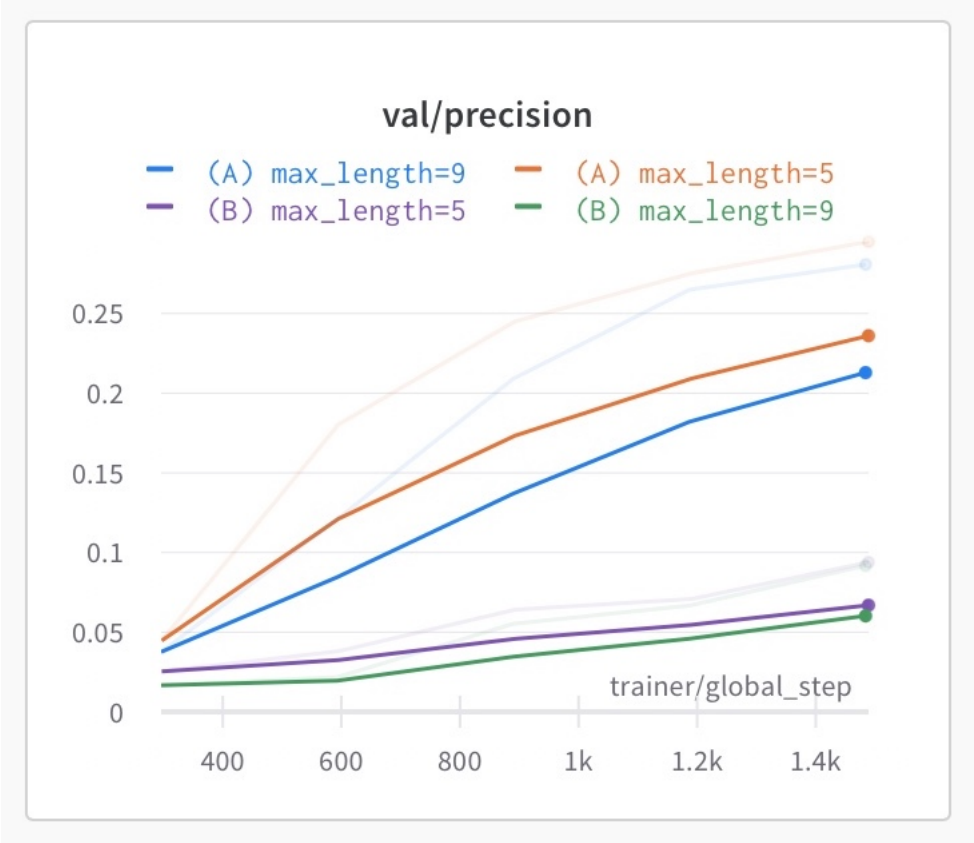


Рисунок 3.6 — Значение метрики precision на валидационной выборке.

3.3 Анализ полученных результатов

Результаты работы алгоритма говорят о том, что такой подход имеет право на существование и показывает приемлемые результаты. Для повышения качества алгоритма далее предлагаются возможные улучшения и расширение пространства признаков.

В таблицах 4 и 5 отражены результаты работы обученной модели на тестовых данных. А также приведены число параметров модели и количество уникальных элементов в выборке.

Таблица 4 — Результаты обученной модели с параметром `max_length=5` на тестовой выборке.

	Конфигурация A	Конфигурация B
Loss	**5.52**	6.59
Precision	**0.29**	0.09
Recall	**0.26**	0.07
F1	**0.27**	0.08
Параметров encoder	401k	121k
Параметров decoder	757k	353k
Грамматических элементов	11k	11k
Типов вершин	80	80

Можно заметить по рисункам 3.7 и 3.8 и таблицам 4 и 5, что увеличение максимально допустимой длины пути (контекстной цепочки) с 5 до 9 не даёт прироста в качестве, зато предобработка данных с параметром max_length=9 занимает значительно больше времени.

Для случайно выбранных фрагментов конфигураций приведены элементы, которые предложены текущей версией контекстной подсказки и предсавтлены на рисунках 3.9, 3.10, 3.11, 3.12, 3.13, 3.14, и топ-5 элементов, которые предсказала обученная модель.

Рисунок 3.9, «Добавить → МассивСвязейПараметровВыбора»:

1. Вставить
2. **Добавить**
3. ЭтотОбъект

Таблица 5 — Результаты обученной модели с параметром `max_length=9` на тестовой выборке.

	Конфигурация A	Конфигурация B
Loss	**5.57**	6.52
Precision	**0.28**	0.09
Recall	**0.25**	0.08
F1	**0.26**	0.08
Параметров encoder	386k	116k
Параметров decoder	733k	342k
Грамматических элементов	11k	11k
Типов вершин	80	80

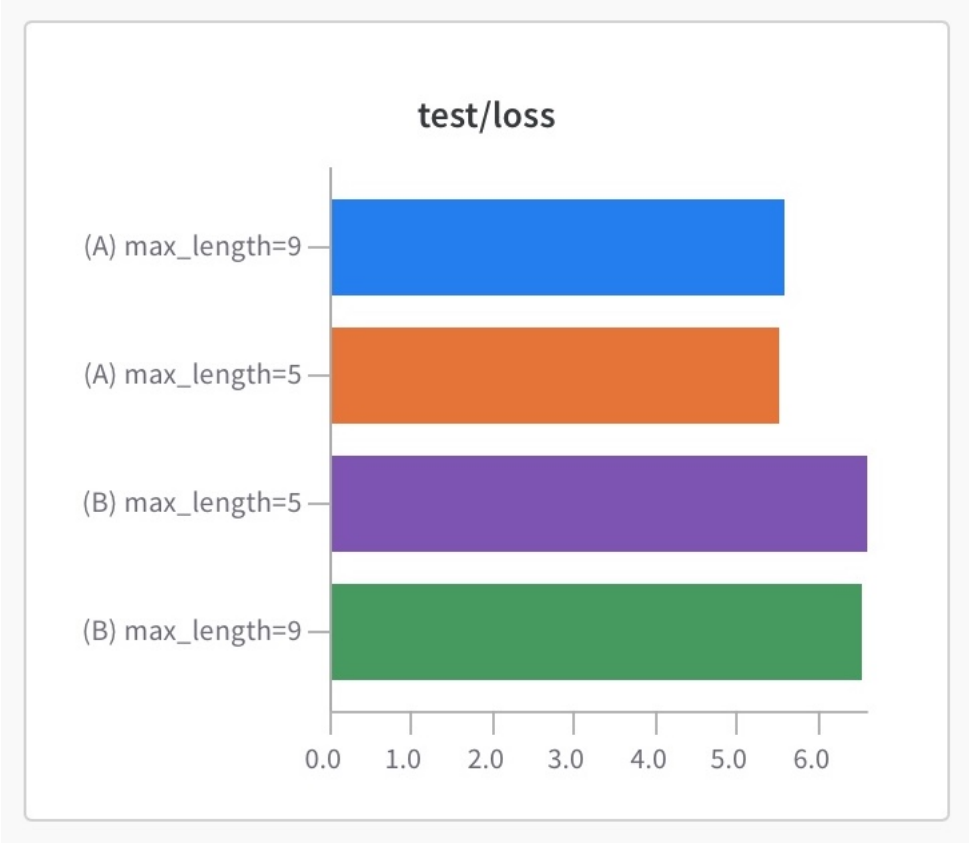


Рисунок 3.7 — Ошибка модели на тестовой выборке.

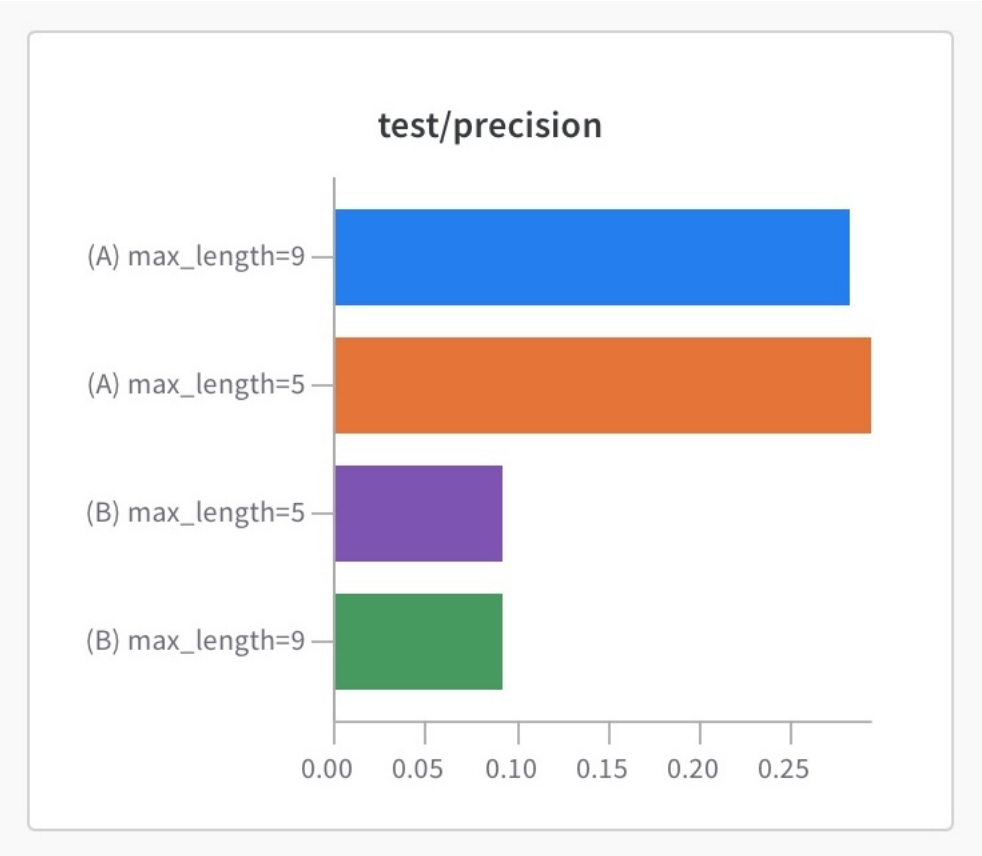


Рисунок 3.8 — Значение метрики precision на тестовой выборке.

4. Параметры
5. Элементы

Рисунок 3.10, «Добавить → МассивУдаляемыхСтрок»:

1. Вставить
2. ЭтотОбъект
3. Текст
4. **Добавить**
5. СлужебныеПроцедурыИФункции

Рисунок 3.11, «Элементы → ГруппаОтбора»:

1. Структура
2. ЭтотОбъект
3. Текст
4. **Элементы**
5. ОписаниеОповещения

Рисунок 3.12, «Ссылка → ЭлементОбъект»:

1. Элементы
2. Параметры
3. НСтр

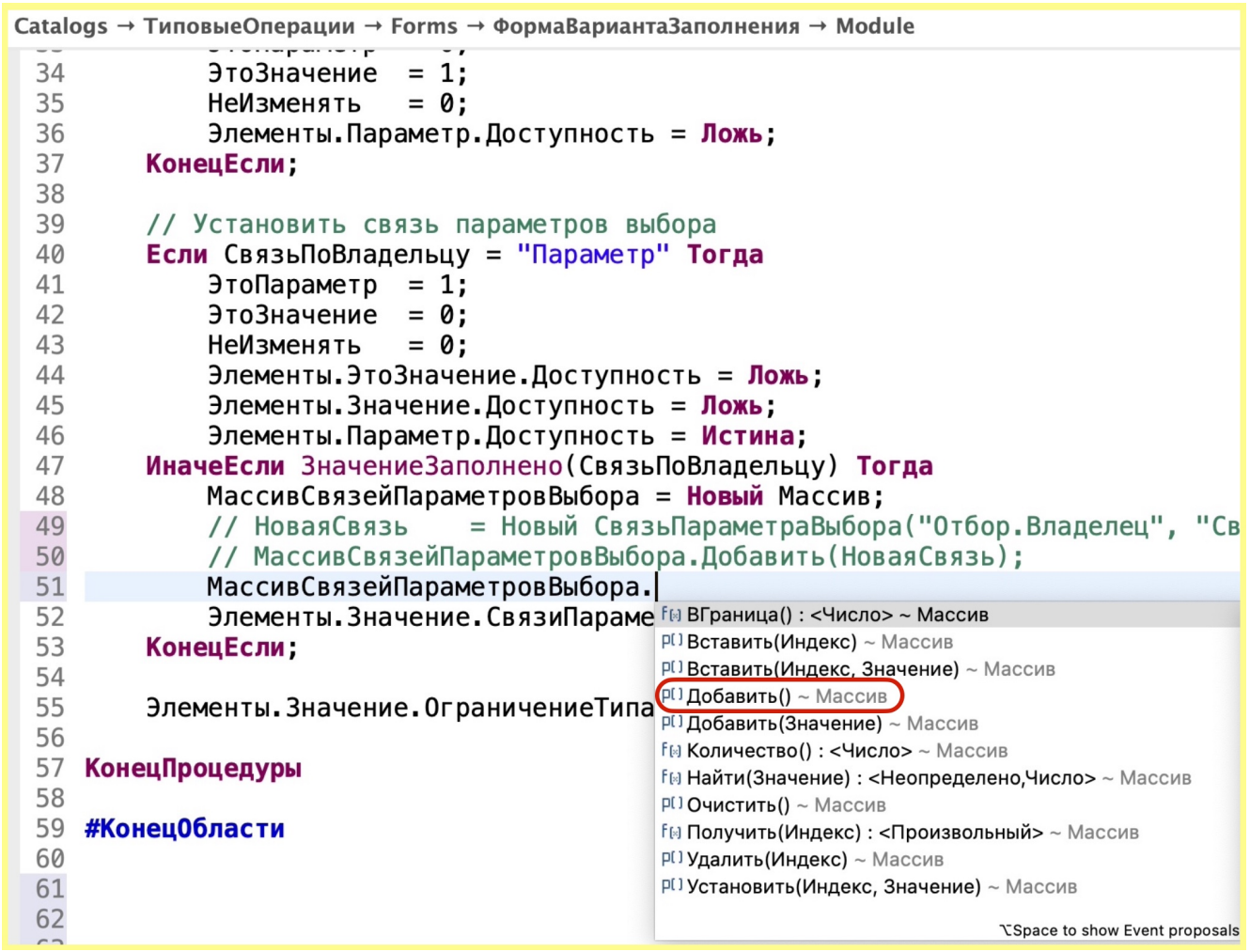


Рисунок 3.9 — Пример выдачи текущей версии контекстной подсказки, встроенной в 1С:EDT.

4. **Ссылка**
5. ОбщегоНазначенияКлиентСервер

Рисунок 3.13, «Свойство → ТекущиеДанные»:

1. Элементы
2. Параметры
3. Вставить
4. **Свойство**
5. НСтр

Рисунок 3.14, «Текст → Запрос»:

1. Структура
2. **Текст**
3. ЭтотОбъект
4. Вставить
5. Запрос

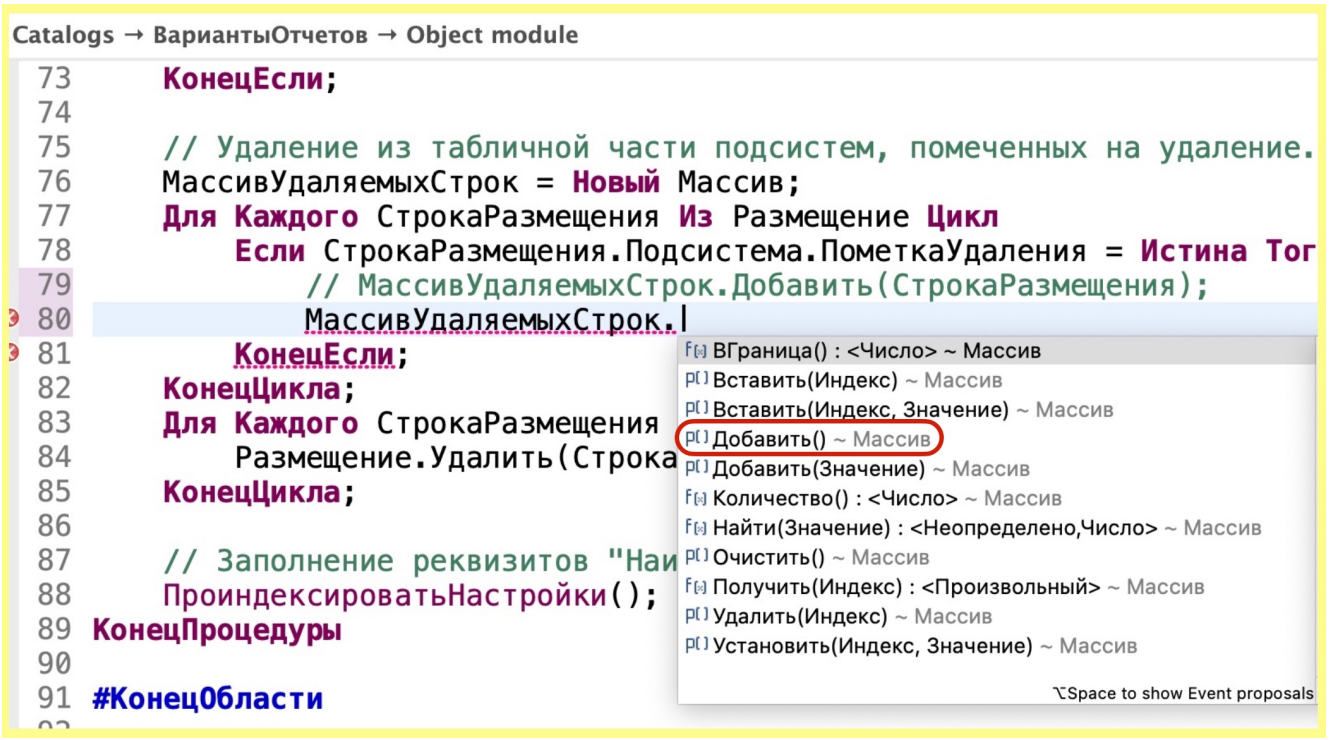


Рисунок 3.10 — Пример выдачи текущей версии контекстной подсказки, встроенной в 1C:EDT.

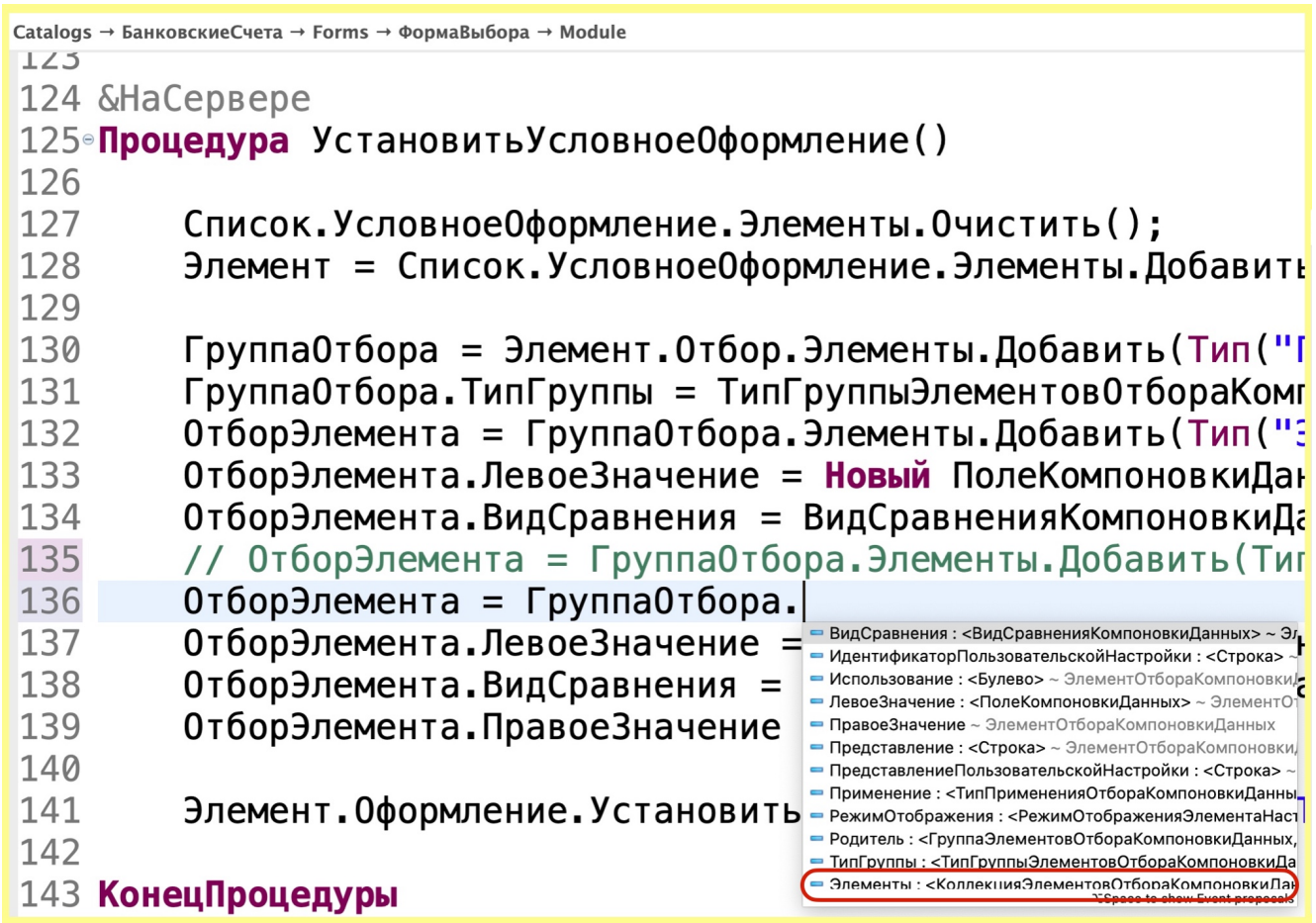


Рисунок 3.11 — Пример выдачи текущей версии контекстной подсказки, встроенной в 1C:EDT.

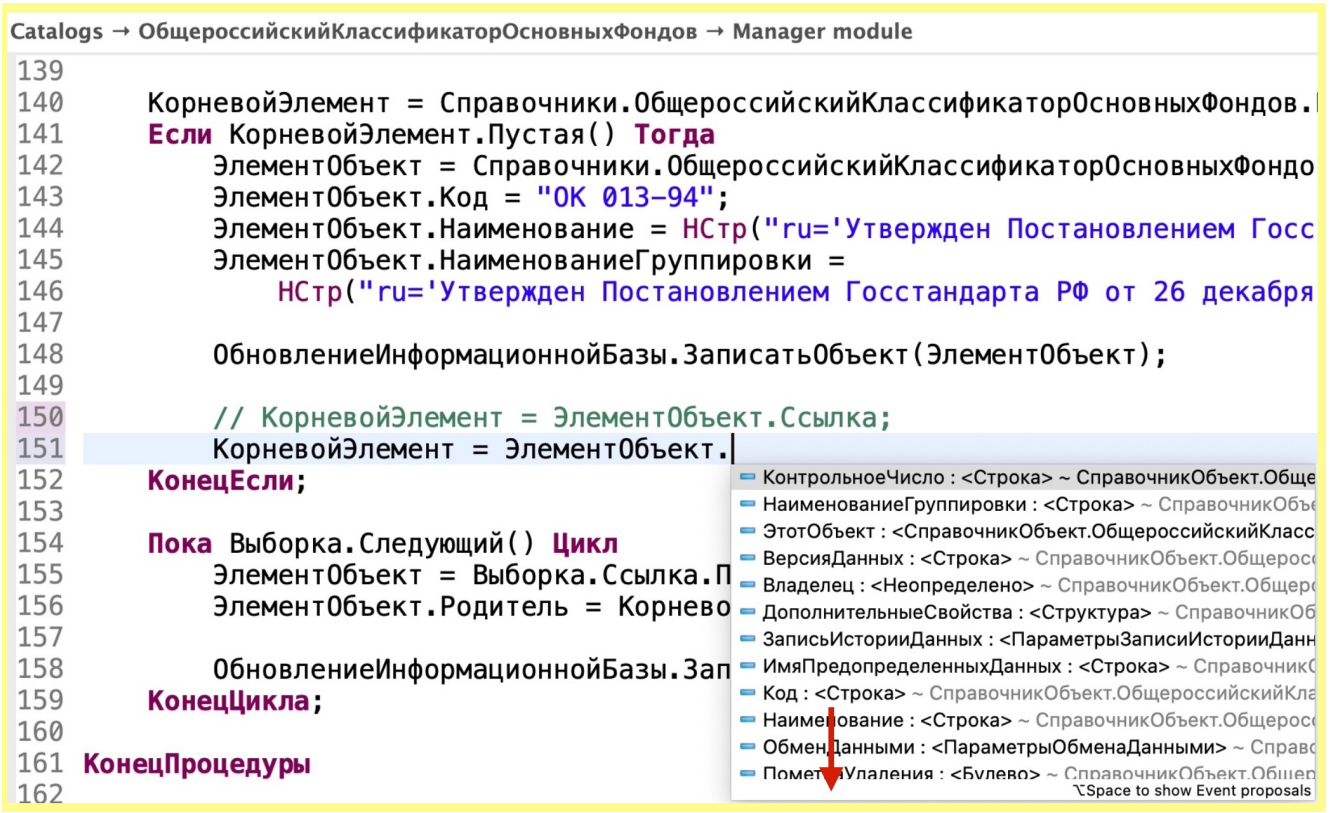


Рисунок 3.12 — Пример выдачи текущей версии контекстной подсказки, встроенной в 1C:EDT.

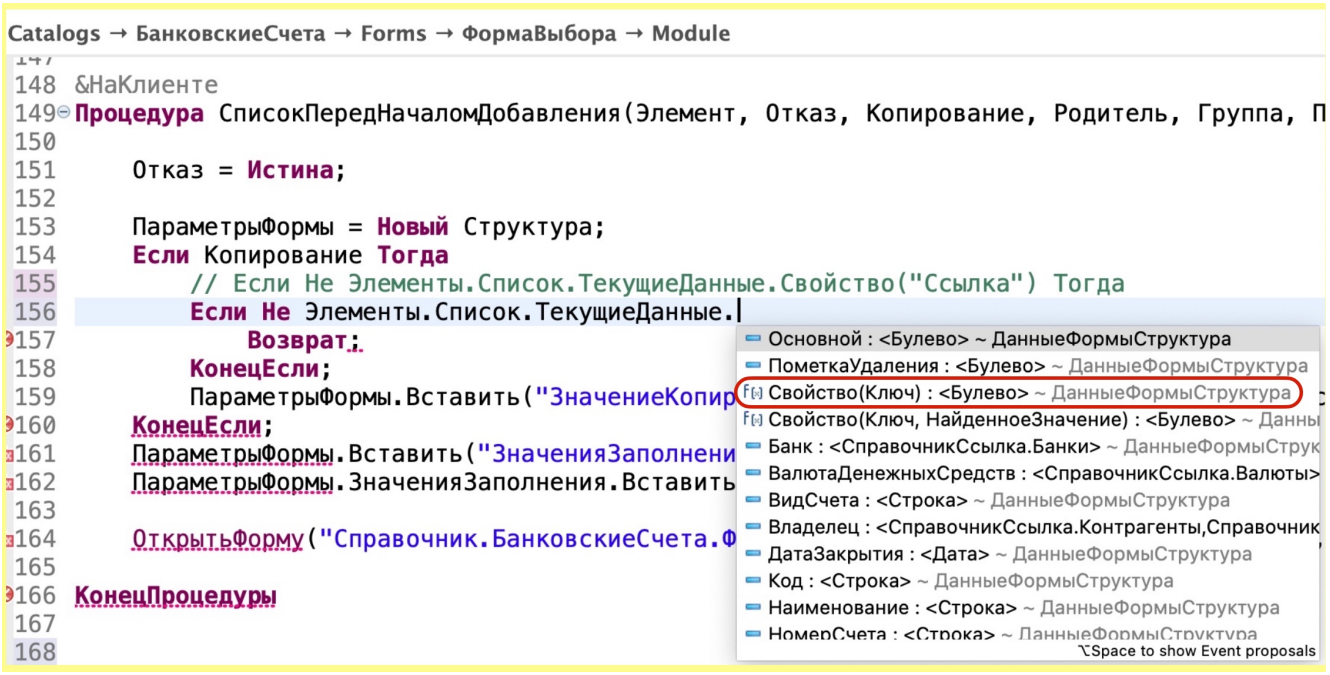


Рисунок 3.13 — Пример выдачи текущей версии контекстной подсказки, встроенной в 1C:EDT.

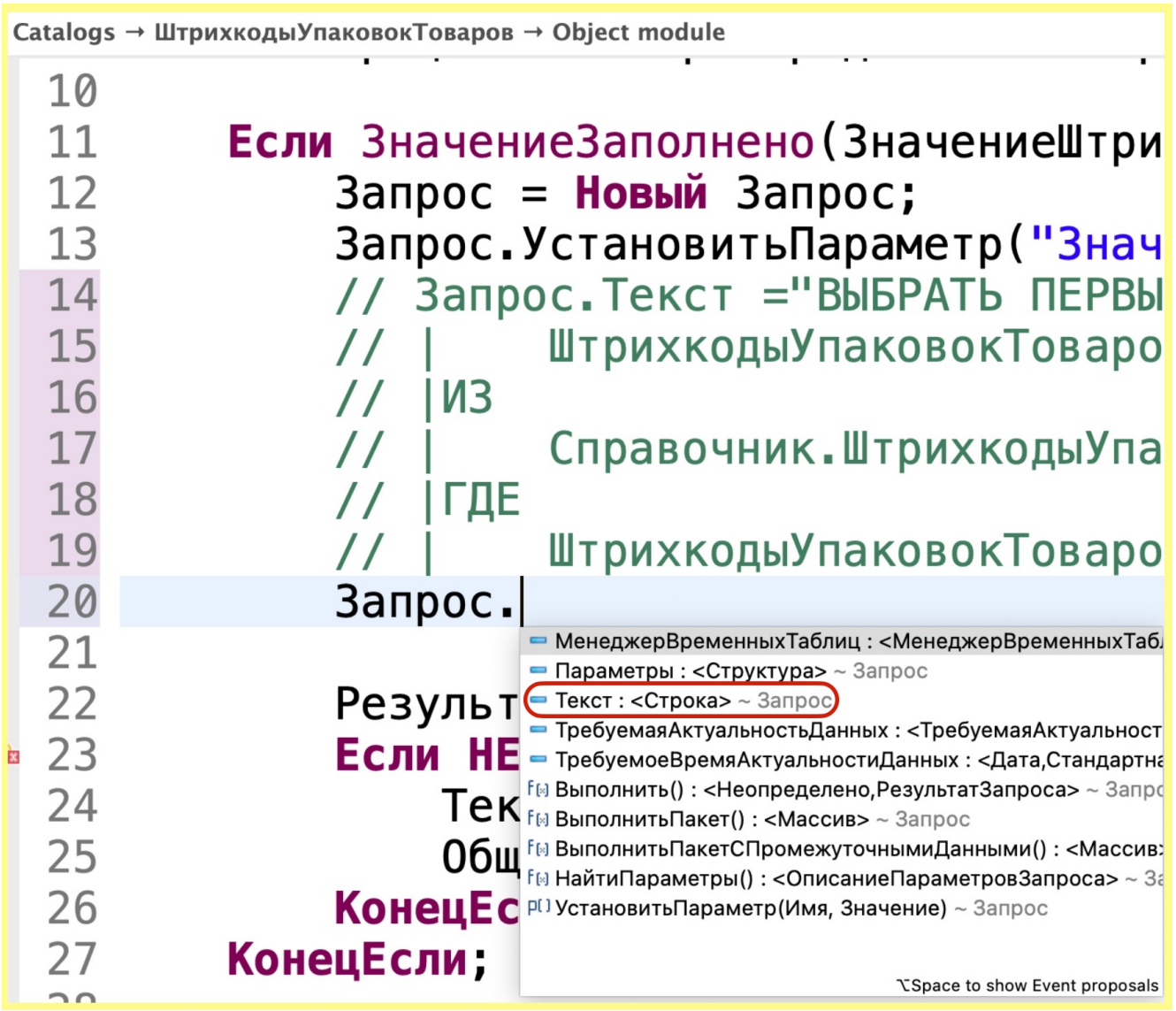


Рисунок 3.14 — Пример выдачи текущей версии контекстной подсказки, встроенной в 1C:EDT.

Глава 4. Заключение

4.1 Выводы

Стоить отметить следующее:

1. Полученные результаты были сделаны только на основании данных из АСД;
2. Это не конечный результат, АСД можно расширить признаками: учитывать типизацию переменных, типы модулей (обработчик, объекты метаданных, метод и тому подобное);
3. Поставленные задачи были достигнуты и предложены способы улучшения алгоритма.

4.2 Дальнейшая работа

Обернуть алгоритм в виде плагина для 1C:EDT.

Это не конечный результат, АСД можно расширить признаками: учитывать типизацию переменных, типы модулей (обработчик, объекты метаданных, метод и тому подобное).

Также стоит ввести функцию близости для вершин деревьев, чтобы можно было оценивать степень похожести (схоже с подходом Maximum likelihood[1]).

4.3 Использованные технологии

1. 1C:EDT - среда разработки прикладных решений на платформе «1С: Предприятие 8»;

[1] https://en.wikipedia.org/wiki/Maximum_likelihood_estimation

2. Eclipse - среда разработки на языке программирования Java и обработки конфигураций встроенном языке платформы «1С: Предприятие 8» посредством фреймворка Xtext;
3. GitHub - система контроля версий;
4. Jupyter - среда для быстрой разработки и прототипирования на языке программирования Python;
5. PyCharm - среда разработки на языке программирования Python;
6. PyTorch - фреймворк для построения и обучения нейронных сетей;
7. PyTorch Lighting - фреймворк для более быстрого и удобного способа создания пайплайна обучения и тестирования нейронных сетей;
8. Weights & Biases - инструмент для мониторинга процесса обучения и тестирования моделей машинного обучения.

Словарь терминов

Абстрактное Синтаксическое Дерево (АСД) : дерево, в котором вершины сопоставлены с операторами языка программирования, а листья - с соответствующими операндами.

Вершина : точка, где могут сходиться или выходить рёбра. Множество вершин графа G обозначается $V(G)$.

Граф : набор вершин и рёбер, где каждое ребро соединяет ровно две вершины. Обозначается $G(V, E)$.

Датасет : набор данных.

Дерево : связный граф без циклов.

Контекстная подсказка : позволяет быстрее вводить релевантные названия переменных и методов, а также избежать ошибок при написании кода программы.

Контент-ассист : см. ”Контекстная подсказка”.

Лист : вершина с единственным ребром.

Ребро : соединяет две вершины графа. Множество рёбер графа G обозначается $E(G)$.

Эмбеддинг : векторное представление слова.

IDE : интегрированная среда разработки; комплекс программных средств, используемый программистами для разработки программного обеспечения.

Список литературы

1. *Raychev, V.* Predicting Program Properties from "Big Code" [Текст] / V. Raychev, M. Vechev, A. Krause // SIGPLAN Not. — New York, NY, USA, 2015. — Янв. — Т. 50, № 1. — С. 111—124. — URL: https://doi.org/10.1145/2775051.2677009.
2. A General Path-Based Representation for Predicting Program Properties [Текст] / U. Alon [и др.] // SIGPLAN Not. — New York, NY, USA, 2018. — Июнь. — Т. 53, № 4. — С. 404—419. — URL: https://doi.org/10.1145/3296979.3192412.
3. The Graph Neural Network Model [Текст] / F. Scarselli [и др.] // IEEE Transactions on Neural Networks. — 2009. — Янв. — Т. 20, № 1. — С. 61—80. — URL: https://doi.org/10.1109/TNN.2008.2005605.
4. *Allamanis, M.* Learning to Represent Programs with Graphs [Текст] / M. Allamanis, M. Brockschmidt, M. Khademi. — 2018. — URL: https://openreview.net/forum?id=BJOFETxR-.
5. Gated Graph Sequence Neural Networks [Текст] / Y. Li [и др.]. — 2015. — URL: http://arxiv.org/abs/1511.05493.
6. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches [Текст] / K. Cho [и др.]. — 2014. — Окт. — URL: https://www.aclweb.org/anthology/W14-4012.
7. code2seq: Generating Sequences from Structured Representations of Code [Текст] / U. Alon [и др.]. — 2019. — URL: https://openreview.net/forum?id=H1gKYo09tX.
8. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation [Текст] / K. Cho [и др.]. — 2014. — arXiv: 1406.1078 [cs.CL].
9. *Sutskever, I.* Sequence to Sequence Learning with Neural Networks [Текст] / I. Sutskever, O. Vinyals, Q. V. Le. — 2014. — URL: https://papers.nips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf.
10. *Luong, T.* Effective Approaches to Attention-based Neural Machine Translation [Текст] / T. Luong, H. Pham, C. D. Manning. — 2015. — Сент. — URL: https://www.aclweb.org/anthology/D15-1166.

11. *Bahdanau, D.* Neural Machine Translation by Jointly Learning to Align and Translate [Текст] / D. Bahdanau, K. Cho, Y. Bengio. — 2016. — arXiv: 1409.0473 [cs.CL].

12. Suggesting Accurate Method and Class Names [Текст] / M. Allamanis [и др.] // Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. — Bergamo, Italy : Association for Computing Machinery, 2015. — С. 38—49. — (ESEC/FSE 2015). — URL: https://doi.org/10.1145/2786805.2786849.

13. *Sennrich, R.* Neural Machine Translation of Rare Words with Subword Units [Текст] / R. Sennrich, B. Haddow, A. Birch. — 2016. — Авг. — URL: https://www.aclweb.org/anthology/P16-1162.

Список рисунков

Список таблиц