# Инструкция по исправлению и улучшению проекта Mermaid NetDocs

**Описание проекта:** Mermaid NetDocs автоматически генерирует **Mermaid**-диаграммы сети из инвентарных данных и (опционально) CLI-конфигураций сетевых устройств[[1]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/README.md#L6-L14). Проект имеет модульную архитектуру:

* **Парсинг CLI (TextFSM)** – модуль textfsm\_loader.py читает конфигурационные файлы (или вывод команд) из data/raw, используя шаблоны TextFSM для разных вендоров[[2]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L10-L18)[[3]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/config/config.json#L26-L34). Результат парсинга – JSON с данными по каждому узлу (интерфейсы, VLAN, соседи и пр.)[[4]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L18-L26).
* **Нормализация инвентаря** – модуль normalize.py объединяет статический инвентарь (из data/input/inventory.json) с данными парсинга, формируя единый граф узлов и связей[[5]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L6-L14). На выходе получается normalized.json с списком nodes и edges[[6]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L10-L18) для рендера.
* **Рендеринг диаграммы** – модуль mermaid\_writer.py берет нормализованный граф и генерирует Mermaid-диаграмму (network.mmd) с учетом правил оформления (направление графа, классы узлов, группировка в подграфы и т.д.)[[7]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L2-L10).

Ниже приведены шаги по пошаговому исправлению программы с помощью Codex GPT, с пояснениями к каждому шагу.

## Шаг 1: Исправить парсинг VLAN из конфигураций Cisco

**Проблема:** Сейчас VLAN-информация не добавляется в инвентарь узлов. В коде нормализации есть логика добавлять VLAN в подписи узлов, но она работает только если в объекте устройства заполнен список vlans[[8]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L266-L274). В текущей реализации эти списки остаются пустыми, так как парсер CLI не извлекает VLAN из конфигураций. В результате узлы не получают меток VLAN в диаграмме.

**Инструкция:** Добавить парсинг VLAN для Cisco (и других вендоров при необходимости):  
- **Шаблон TextFSM для VLAN:** Создать шаблон (например, show\_vlan.template для Cisco IOS) или регулярное выражение для извлечения VLAN ID и имени VLAN из конфигурации. Для Cisco IOS можно парсить секцию vlan в выводе show running-config или использовать команду show vlan brief. Убедиться, что шаблон сохраняет ID VLAN и имя (если есть) в структуру данных.  
- **Обновление Inventory.vlans:** В модуле парсинга CLI, после обработки каждого устройства, добавлять найденные VLAN в соответствующий объект устройства. Например, расширить класс Device (в normalize.py) или аналогичный DTO, чтобы включить список VLAN. При парсинге записывать так:

device\_obj.vlans.append({"id": <VLAN\_ID>, "name": <VLAN\_NAME>})

или, если VLAN уже выделены в parsed\_data, скопировать их в Inventory. Убедиться, что для каждого устройства Cisco IOS (и аналогично для других вендоров, если возможно) заполняется device.vlans.  
- **Валидация:** После внесения изменений запустить нормализацию. В normalized.json у узлов должны появиться VLAN. В network.mmd эти VLAN отобразятся как дополнительные строки в узле (до 5 VLAN, остальные сворачиваются)[[9]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L270-L278). Например, узел Cisco с VLAN17 и VLAN91 должен иметь в Mermaid-лейбле строку VLAN 17 — NameOfVLAN17 и т.д. Проверьте, что длина подписей VLAN ограничена 64 символами (логика truncate)[[10]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L270-L274), чтобы диаграмма оставалась читаемой. Добавьте юнит-тест: подайте фрагмент конфигурации Cisco с VLAN и проверьте, что после парсинга объект устройства содержит ожидаемые VLAN и что функция нормализации добавляет их в labels.extra.

## Шаг 2: Добавить поддержку особенностей Mermaid-диаграмм

**Проблема:** Генератор диаграммы не полностью соответствует требованиям стиля Mermaid. Необходимо убедиться, что диаграмма оформляется по правилам из standards/mermaid\_style.md. Есть два важных момента: (1) флаг **htmlLabels** должен быть выключен (false) для корректной работы переносов строк, (2) узлы особых типов (например, Wi-Fi точки) нужно выделять особым стилем.

**Инструкция:** Подправить генерацию Mermaid-диаграммы согласно гайдам:  
- **Отключить htmlLabels:** В функции \_header() сейчас жестко задано "htmlLabels": true[[11]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L182-L185), что противоречит стилевому руководству (требуется false[[12]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/standards/mermaid_style.md#L20-L26)). Исправьте строку и поставьте "htmlLabels": false. Это позволит использованию тегов <br/> для переносов строк (уже экранируются функцией esc()[[13]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L116-L124)) работать правильно без HTML-лейблов. После изменения убедитесь, что многострочные подписи (hostname, IP, модель на разных строках) отображаются корректно.  
- **Подграфы по площадкам и ролям:** Удостоверьтесь, что параметр use\_subgraphs из конфига учитывается. В коде это реализовано: при use\_subgraphs=true узлы группируются по site -> role[[14]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L236-L244)[[15]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L206-L215). Однако проверьте формат меток подграфов. Сейчас id подграфа устанавливается равным названию площадки (site)[[16]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L207-L215). Если нужно задать кастомный отображаемый заголовок, можно использовать синтаксис subgraph ID["Label"]. Например, чтобы вместо кода площадки INA39 показать описание "39 - 3-й этаж АБК...", можно хранить в инвентаре человекочитаемое название и выводить его в скобках. При необходимости, измените вывод: lines.append(f'subgraph {site}["{display\_name}"]'). Обеспечьте экранирование кавычек в названии (через esc()).  
- **Классы узлов (стили):** В Mermaid Style Guide перечислены классы core, dist, access, wan, dmz, mgmt[[17]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L12-L16), а также упоминается класс для Wi-Fi. В коде уже заданы определения classDef для core,...,mgmt и wifi[[18]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L157-L165). Однако сейчас присвоение класса узлу происходит на основе поля Device.role (или class) – всё, что не распознано, становится "access" по умолчанию[[19]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L136-L140). Нужно добавить поддержку Wi-Fi и других специальных классов:  
- **Wi-Fi узлы:** Определить критерий, по которому устройство считается беспроводной точкой (например, device.vendor == "mikrotik" для Wi-Fi контроллеров, либо имя/модель содержит подсказку – в примерах узлы названы ...AIR... для Wi-Fi). Реализовать присвоение класса wifi: например, если Device.role == "wifi" или модель содержит "AP", то устанавливать device.class = "wifi". Тогда при рендере узел получит класс wifi и стиль (заливка зеленоватым, пунктирная рамка иконки)[[20]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L159-L167).  
- **Неизвестные/новые классы:** Если в будущем появятся новые типы узлов (например, firewall, storage и т.д.), продумайте расширяемость. Можно хранить список разрешенных классов в конфиге (config.json), например render.classes, и валидировать роль узла против него. Сейчас конфиг содержит список классов[[21]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/config/config.json#L24-L32), но wifi там отсутствует. Добавьте wifi в render.classes конфигурации, чтобы следовать тому же паттерну.  
- **Проверка:** Сгенерируйте диаграмму на тестовых данных. Убедитесь, что в начале диаграммы присутствует заголовок с flowchart LR и настройками (theme, curve)[[22]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L180-L188), а также **внизу** диаграммы перечислены определения классов (classDef)[[18]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L157-L165) – это проверяется в тестах[[23]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L60-L67). Диаграмма должна корректно отображать подграфы (блоки сгруппированных узлов) по площадкам, а Wi-Fi точки – отличаться стилем. Если возможно, визуально просмотрите результат в Mermaid-рендерере. Кроме того, добавьте тест, проверяющий, что при use\_subgraphs=true текст диаграммы содержит строки subgraph с именами площадок из инвентаря.

## Шаг 3: Исправить логику построения топологии (связей между устройствами)

**Проблема:** Текущая программа не формирует связи (ребра графа) между узлами, если информация о соседях не указана явно в инвентаре. В ходе парсинга CLI информация о линках (CDP/LLDP соседи, подключения интерфейсов) не переносится в структуру Inventory.interfaces. В результате после нормализации граф пуст – нет рёбер (edges=0)[[24]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/data/output/logs/scan_test.log#L14-L20), и на диаграмме все узлы изолированы. Это ключевой недостаток: **топология не строится автоматически**.

**Инструкция:** Реализовать заполнение связей между устройствами на основе данных из конфигураций:  
- **Парсинг соседей:** Добавить шаблоны и логику для сбора соседей по интерфейсам. Например, для Cisco IOS – шаблон TextFSM для команды show cdp neighbors detail или show lldp neighbors (универсальнее). Аналогично для других вендоров (Huawei – display lldp neighbor, MikroTik – neighbors print, и т.д.) с помощью соответствующих шаблонов. Цель – получить для каждого устройства список интерфейсов и на каждом интерфейсе, если обнаружен сосед, имя устройства-соседа и порт.  
- **Обогащение Inventory.interfaces:** После парсинга необходимо перенести полученные связи в инвентарь перед фазой нормализации:  
- Расширить структуру Device.interfaces (в normalize.py) – каждый интерфейс может быть представлен словарем с полями: имя интерфейса (name), скорость/тип (speed, если известно), описание (desc), и вложенным объектом peer с данными соседа[[25]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L231-L240). Формат уже задан: peer: {"hostname": ..., "interface": ...}[[26]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L233-L241).  
- В модуле textfsm\_loader.py после парсинга CLI сформировать список интерфейсов для каждого устройства. Для этого можно: 1. Сопоставить вывод команды LLDP/CDP с устройствами. Например, после парсинга show lldp neighbors получить список записей: локальный интерфейс, имя соседа, порт соседа. 2. Пройтись по этим записям и добавить соответствующие структуры в Inventory.devices[hostname].interfaces. Если устройство или интерфейс уже существуют в инвентаре, обновить их; если нет – создать новую запись. Например:

dev = inventory.devices[local\_host]  
dev.interfaces.append({  
 "name": local\_intf,  
 "peer": {"hostname": neighbor\_name, "interface": neighbor\_port},  
 "desc": "Link to " + neighbor\_name # опционально  
})

3. При добавлении соседа, проверить: если сосед neighbor\_name отсутствует в inventory.devices, **создать новый Device** для него (с hostname=neighbor\_name и минимальными полями). Это предотвратит ситуацию, когда ребро указывает на несуществующий узел[[27]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L259-L262). Новый узел можно пометить, например, role="access" и vendor=None (или "unknown"), чтобы потом он появился как узел на диаграмме.  
- **Дублирование и направление:** Чтобы избежать дублирования связей (например, если DeviceA видит DeviceB и наоборот), можно вводить связь только при парсинге одного из них. CDP обычно односторонний (Cisco видит соседа с именем, а сосед – нет, если не Cisco). LLDP двусторонний, но у каждого устройства будет запись о соседе, что может привести к добавлению двух записей (A->B и B->A). Можно решить это так: вводить связь только если имя локального узла лексически меньше имени соседа (чтобы консистентно выбирать одну сторону) или помечать добавленные пары и пропускать дубль. Простое решение: Normalizer при обходе интерфейсов уже создаёт все ребра, дубли не соединит, а просто выдаст две однонаправленные связи (которые Mermaid всё равно отобразит как единый без стрелок). Однако лучше отфильтровать дубли заранее.  
- **Speed/VLAN на ребрах:** Если есть возможность, парсите информацию о типе подключения. В Cisco можно получить, является ли интерфейс транком и какие VLAN разрешены (например, командой show interfaces switchport или в выводе CDP detail). В Mermaid-диаграмме из исходных данных видно, что на ребрах подписаны VLAN для транков (пример: "trunk 1,17,30–34..."). Добиться такого вывода можно двумя способами: 1. Включить VLAN в поле desc или отдельное поле при формировании ребра. Сейчас код берёт поля name (интерфейс), speed и desc для метки ребра[[28]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L237-L245). Можно поместить строку "trunk X,Y..." в desc связи. Например, если интерфейс является trunk, заполнить itf.desc = "trunk " + vlan\_list. 2. Либо расширить формат edges, добавив поле vlans (оно уже есть в структуре edges[[29]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L34-L40)), а в функции build\_edge\_line включить VLANы в подпись. Однако это потребует менять генератор строк. Более просто – вариант (1): добавьте VLANы к описанию интерфейса в Inventory.  
- **Валидация:** После внедрения механизма связей выполните полный прогон конвейера на тестовом инвентаре. Ожидается, что в логе нормализации появятся узлы и рёбра (например, nodes=5, edges=4) вместо нулей[[24]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/data/output/logs/scan_test.log#L14-L20). Проверьте содержимое normalized.json: он должен содержать массив edges с заполненными src, dst и метками. Например:

"edges": [  
 {  
 "src": "SW1", "src\_intf": "Gi1/0/1",  
 "dst": "SW2", "dst\_intf": "Gi0/48",  
 "desc": "trunk 1,17,50",   
 "label": "Gi1/0/1 | trunk 1,17,50",  
 "layer": "L2", "vlans": [1,17,50]  
 },  
 ...  
]

Метка (label) формируется автоматически из src\_intf | speed | desc[[28]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L237-L245), проверьте, что она не превышает 80 символов (лишнее обрежется)[[30]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L29-L37). На диаграмме Mermaid убедитесь, что появляются линии между соответствующими узлами с правильными подписями портов с двух сторон. Добавьте тест, который создаёт минимальный инвентарь из двух устройств с искусственно заданными peer-интерфейсами, запускает нормализацию и проверяет, что образуется ровно одно ребро между ними. Также логично протестировать случай, когда у устройства есть ссылка на несуществующий узел – программа должна создать узел-виртуалку или по крайней мере вывести предупреждение, но не падать (такой случай уже обрабатывался предупреждением peer\_not\_in\_inventory[[31]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L258-L262), но после ваших изменений неизвестных соседей вообще лучше добавлять как узлы).

## Шаг 4: Улучшить структуру кода и читаемость

**Проблема:** Хотя код уже достаточно структурирован, есть моменты, затрудняющие поддержку: дублирование определений, несогласованность форматов, отсутствие ясности в некоторых местах. Например, класс Inventory определён дважды – отдельно в парсере TextFSM и в нормализаторе[[32]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L110-L118)[[33]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L161-L169) – с разными полями. Кроме того, часть функционала (слияние инвентаря с CLI) реализована скриптом pipeline.py, что может вызывать несоответствия, как показал конфликт JSON vs YAML в тестах[[34]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L50-L58).

**Инструкция:** Провести рефакторинг для повышения ясности:  
- **Унификация модели инвентаря:** Рассмотрите объединение классов Inventory из разных модулей в одну структуру данных. Например, можно создать единый dataclass Inventory (в отдельном файле или модуле), который будет содержать все необходимые поля: список устройств (с их основными атрибутами) и, возможно, сразу маппинг hostname -> CLI файлы. Тогда модуль textfsm\_loader будет наполнять этот объект (добавлять устройства, интерфейсы, VLAN, и т.п.), а модуль normalize сможет его напрямую использовать. Если полностью объединить не получается, то **разграничьте чётче ответственность**: - textfsm\_loader.Inventory – переименуйте, например, в RawInventory или ParsedInventory, чтобы указать, что это инвентарь на этапе парсинга CLI. - normalize.Inventory – можно назвать TopologyInventory или MergedInventory, т.е. итоговый.  
Обновите места использования и комментарии, чтобы разработчикам было понятно отличие.  
- **Формат inventory-файлов:** Убедитесь, что проект использует единый формат инвентаря. Первоначально, судя по комментариям, поддерживался YAML, но теперь всюду JSON (см. пометки *YAML больше не используется* в коде[[35]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L54-L61)[[36]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L58-L65)). Однако тесты ссылаются на inventory.yml[[34]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L50-L58), что может быть остатком прошлого. Приведите все примеры и тесты к использованию inventory.json для согласованности. Если YAML больше не нужен, удалите или исправьте соответствующие упоминания в README и комментариях. Это предотвратит путаницу.  
- **Слияние данных (pipeline):** Пересмотрите реализацию шага слияния инвентаря (функция step\_merge\_inventory в pipeline.py). Сейчас она добавляет устройства из cli/\*.json в базовый список[[37]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/pipeline.py#L180-L189)[[38]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/pipeline.py#L191-L200), но не объединяет подробности (интерфейсы, IP) устройств. Возможно, после внедрения вышеупомянутых шагов, логика слияния упростится: если вы сразу заполняете основной Inventory данными CLI, то отдельный JSON inventory.merged.json может оказаться избыточным. Можно сделать так: если inventory.json существует, загрузить его и передать в TextFSMParser для дополнения; если нет – TextFSMParser сам создаст devices (как он уже умеет делать при пустом инвентаре)[[39]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L239-L248). Тогда step\_merge\_inventory будет просто сохранять объединённый инвентарь.  
- **Логирование и отладка:** Сохраните или улучшите существующие логи. Например, сейчас при парсинге выводится сколько шаблонов загружено для каждого вендора[[40]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/data/output/logs/scan_test.log#L2-L10) и предупреждения для неизвестных вендоров[[41]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/data/output/logs/scan_test.log#L4-L11) – это полезно. Можно расширить: логировать, сколько соседей найдено, сколько VLAN добавлено и т.п. Это поможет верифицировать работу на реальных данных.  
- **Очистка неиспользуемого кода:** Если в ходе рефакторинга какие-то части устареют (например, файлы или опции, оставшиеся от старой реализации), удалите их. Проект должен быть структурирован так: конфигурация (config.json), входные данные (inventory.json, raw CLI), выходные данные (parsed, normalized, диаграмма), src/ модули, tests/. Проверьте, что директории соответствуют README. Обновите README.md после правок: опишите новые шаги, форматы, чтобы другие разработчики или Codex GPT могли легче понять проект.

## Шаг 5: Улучшить наименование функций, обработку ошибок и структуры данных

**Проблема:** Некоторые аспекты кодовой базы можно сделать более понятными и надёжными. Например, именование переменных типа d, data, fp в функции слияния неочевидно[[42]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/pipeline.py#L182-L190), а обработка ошибок в точках входа просто пробрасывает исключение вверх (что приводит к *traceback*, а не аккуратному сообщению пользователю)[[43]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L292-L299). Необходимо внедрить соглашения по именам и более дружелюбную обработку ошибок.

**Инструкция:** Внести следующие улучшения:  
- **Имена функций и переменных:** Следуйте **единому стилю именования** – в проекте уже принят стиль lower\_case\_with\_underscores для функций и CamelCase для классов. Проверьте, что новые функции (например, обработчики VLAN или соседей) названы глагольными названиями, отражающими действие (например, parse\_vlan\_info, add\_neighbor\_links). Избегайте одно-буквенных переменных за исключением простых счётчиков. Например, в step\_merge\_inventory вместо d используйте host\_data, fp → file\_path, devs → devices\_list для ясности. Переименуйте внутренние поля, где нужно, чтобы не возникало путаницы: например, в JSON после парсинга CLI используется ключ "hosts"[[44]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L16-L24) – можно преобразовать его в "devices" при объединении, чтобы формат был единообразным.  
- **Обработка ошибок и валидация:** Сделайте работу утилиты устойчивее к ошибкам ввода. Некоторые проверки уже есть – например, предупреждения, если не удалось прочитать файл или не найден шаблон[[45]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L222-L230)[[46]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L278-L286). Расширьте их: - Если конфигурационный файл config.json не найден или повреждён, выводите понятное сообщение и останавливайте программу с кодом ошибки, без вывода трассировки Python. Например, оберните загрузку Config.load в try/except и сделайте sys.exit("Config file not found or invalid"). - Аналогично с inventory.json: тесты ожидают, что если файла нет, будет AssertionError с понятным текстом[[47]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L21-L24). Можно явно проверять наличие важных входов и выдавать log.error перед исключением. - В MermaidWriter.render() вместо голого raise при ошибке рендера можно вернуть строку с сообщением об ошибке, чтобы хотя бы пустой файл не выводился молча. Либо логировать и продолжать – решите, что лучше для конечного пользователя.  
- **Работа с неизвестными данными:** Логи указывают, что встречались неучтённые вендоры: Fortigate, Zyxel, Brocade и др. Сейчас программа просто предупреждает о них[[41]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/data/output/logs/scan_test.log#L4-L11). Желательно предусмотреть хотя бы дефолтную обработку: например, все неизвестные вендоры помечать как "unknown" с использованием шаблонов по умолчанию (если есть) или пропускать парсинг CLI для них, но при этом включать в итоговую диаграмму (как узлы без подробной информации). Можно добавить в config.json секцию "default\_vendor": "cisco\_ios" или "unknown\_vendor": { ... } с шаблоном, который матчится на любой текст – чтобы, например, просто вытаскивать hostname и интерфейсы link-требований. Это не обязательно решать полностью, но важно **не допустить падения** программы, если встречается устройство, не описанное в config. Сейчас падения нет – идет предупреждение – это хорошо; оставьте эту логику, но документируйте её и, возможно, объедините повторяющиеся предупреждения (чтобы не захламлять вывод).  
- **Структуры данных:** По возможности, сделайте структуры более явными. Сейчас, например, parsed\_textfsm.json содержит довольно вложенную структуру. Можно дать пример (в комментарии или документации) формата этого JSON, чтобы разработчики знали, где искать нужные данные. Или рассмотрите введение типов (с помощью TypedDict или dataclasses) для этих промежуточных структур, чтобы IDE/lинтер могли отловить несоответствия. Также, после рефакторинга Inventory, убедитесь, что в тестах используются правильные поля. Например, если теперь Inventory.devices – словарь (ключи – hostname), то тесты должны обращаться к нему, а не предполагать список. Обновите тесты согласно новым структурам, чтобы они проходили.

После этих улучшений кодовая база станет понятнее: именованные функции облегчат навигацию, а надёжная обработка ошибок позволит запускать утилиту в различных сценариях без неожиданных сбоев. Добавьте соответствующие проверки: запустите pytest и убедитесь, что все тесты проходят. Введите новые тест-кейсы на обработку ошибок (например, отсутствие файла, неизвестный вендор – ожидается предупреждение, а не исключение).

## Шаг 6: Добавить тесты и примеры для валидации работы

После внесения изменений нужно убедиться, что программа работает правильно на разных входных данных. Разработайте набор тестов и примеров использования, которые проверят основные сценарии:

* **Юнит-тесты парсеров:** Для каждого нового парсера TextFSM (VLAN, LLDP/CDP соседи и др.) создайте небольшой пример входного текста и ожидаемый результат. Например, для VLAN: вход – кусок конфигурации с vlan 17 name Users, выход – объект с {"id": 17, "name": "Users"}. Для LLDP: вход – упрощённый вывод show lldp neighbors (можно замокать часть), выход – список соседей. Эти тесты гарантируют, что шаблоны корректно извлекают данные.
* **Тест нормализации и рендера (end-to-end, smoke test):** Уже есть тесты test\_normalize\_smoke и test\_render\_smoke[[48]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L20-L28)[[49]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L45-L54). Обновите их под новую логику:
* test\_normalize\_smoke: подготовьте минимальный inventory.json (или используйте собранный inventory.merged.json после парсинга) с 1-2 устройствами, включая у них интерфейсы и VLAN. Запустите Normalizer и проверьте, что на выходе хотя бы 1 узел и 1 ребро[[50]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L28-L36). Добавьте проверку, что warnings список содержит ожидаемые предупреждения (или пуст, если всё в порядке). Например, если у нас специально не указали peer для одного интерфейса, Normalizer должен выдать edge\_without\_peer warning[[51]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L255-L262) – поймайте и убедитесь, что он там.
* test\_render\_smoke: сгенерируйте normalized.json (в самом тесте, как сейчас делается через NormInventory/Nomalizer) и передайте в MermaidWriter. Проверьте, что текст диаграммы содержит ключевые элементы: flowchart LR, определения классов (core, access, wifi)[[23]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L60-L67), и все узлы из norm.nodes[[52]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L64-L67). Дополнительно, проверьте наличие подграфов: если use\_subgraphs=True в config, то в выводе должны быть строки вида subgraph. Если Wi-Fi узел есть, должна появиться строка class WiFiNode wifi (или аналогично) в тексте диаграммы.
* **Интеграционный тест на топологию:** Соберите небольшой искусственный пример: два коммутатора разных вендоров, соединённых друг с другом, с несколькими VLAN. Например, устройство A (Cisco) с VLAN10 (Users) и VLAN20 (Servers), устройство B (D-Link) с VLAN10, соединены через порты Gi1/0/1 – 1/0/24. Создайте файлы конфигурации (или нужные выжимки CLI) для них и инвентарь JSON с минимальными данными (hostname, vendor, role, site). Запустите полный конвейер (можно через вызов pipeline.main с параметрами) и получите network.mmd. Проверьте, что в network.mmd присутствуют: две ноды с именами A и B, связь между ними с правильными портами, и что у каждой ноды указаны VLAN (в extra-лейблах, не более 5, остальные должны быть свёрнуты в +N VLANs если больше)[[9]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L270-L278). Этот тест эмулирует реальный случай использования и подтверждает работоспособность всей цепочки.
* **Примеры в документации:** Добавьте в README.md или отдельный файл examples/ пример использования. Например: *«Для запуска на своих данных: заполните data/input/inventory.json устройствами (hostname, site, role, mgmt\_ip и пр.), сохраните сырой вывод команд (например, show run, show lldp neighbors) в data/raw с именами файлов по IP или hostname. Затем выполните python -m src.pipeline. В результате получите файл data/output/network.mmd. Его можно просмотреть через Mermaid live editor.»* Также приведите фрагмент полученной Mermaid-диаграммы в Markdown (как пример). Это поможет пользователям и будет служить неявной проверкой: следуя инструкции, они должны получить корректный результат.

После добавления тестов запустите их: все должны проходить. Исправьте любые выявленные несоответствия. В результате этих шагов приложение Mermaid NetDocs будет корректно парсить входные данные, строить полную топологию сети и генерировать аккуратную диаграмму. Codex GPT, следуя этой инструкции, должен поэтапно внести изменения в код, после каждого шага убедиться, что новые и существующие тесты проходят, и что функциональность соответствует ожидаемой. Такой поэтапный подход минимизирует вероятность регрессий и обеспечит улучшение качества проекта.

[[1]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/README.md#L6-L14) README.md

<https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/README.md>

[[2]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L10-L18) [[4]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L18-L26) [[32]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L110-L118) [[35]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L54-L61) [[39]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L239-L248) [[44]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L16-L24) [[45]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L222-L230) [[46]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py#L278-L286) textfsm\_loader.py

<https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/textfsm_loader.py>

[[3]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/config/config.json#L26-L34) [[21]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/config/config.json#L24-L32) config.json

<https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/config/config.json>

[[5]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L6-L14) [[6]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L10-L18) [[8]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L266-L274) [[9]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L270-L278) [[10]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L270-L274) [[25]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L231-L240) [[26]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L233-L241) [[27]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L259-L262) [[28]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L237-L245) [[29]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L34-L40) [[31]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L258-L262) [[33]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L161-L169) [[36]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L58-L65) [[51]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py#L255-L262) normalize.py

<https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/parsers/normalize.py>

[[7]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L2-L10) [[11]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L182-L185) [[13]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L116-L124) [[14]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L236-L244) [[15]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L206-L215) [[16]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L207-L215) [[17]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L12-L16) [[18]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L157-L165) [[19]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L136-L140) [[20]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L159-L167) [[22]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L180-L188) [[43]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py#L292-L299) mermaid\_writer.py

<https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/renderers/mermaid_writer.py>

[[12]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/standards/mermaid_style.md#L20-L26) mermaid\_style.md

<https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/standards/mermaid_style.md>

[[23]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L60-L67) [[30]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L29-L37) [[34]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L50-L58) [[47]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L21-L24) [[48]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L20-L28) [[49]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L45-L54) [[50]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L28-L36) [[52]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py#L64-L67) test\_normalize\_and\_render\_smoke.py

<https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/tests/test_normalize_and_render_smoke.py>

[[24]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/data/output/logs/scan_test.log#L14-L20) [[40]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/data/output/logs/scan_test.log#L2-L10) [[41]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/data/output/logs/scan_test.log#L4-L11) scan\_test.log

<https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/data/output/logs/scan_test.log>

[[37]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/pipeline.py#L180-L189) [[38]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/pipeline.py#L191-L200) [[42]](https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/pipeline.py#L182-L190) pipeline.py

<https://github.com/mrsokol2552-source/network/blob/e687b8dfecca75f3442f1f7dcfa5614426d23151/src/pipeline.py>