Relatório Consolidado: Aprendizados e Code Review

Erick Bonruque

December 2, 2025

1 Aprendizados Consolidados e Votação

Este documento consolida os aprendizados obtidos a partir da revisão técnica cruzada dos notebooks da FASE 2, conforme o objetivo de aprofundar o entendimento sobre o pipeline SINKT.

1.1 O que aprendi analisando o código dos outros

A análise comparativa dos diferentes notebooks foi extremamente valiosa e revelou uma diversidade de abordagens para resolver o mesmo problema, cada uma com seus pontos fortes. Os principais aprendizados foram:

- **A Importância da Semântica nos Embeddings (TIEnc):** Meu notebook inicial utilizava embeddings aleatórios, que não carregam nenhum significado. Ao analisar o trabalho de **Lucas Parteka**, ficou evidente que o uso de um modelo de linguagem pré-treinado como o **BERT** para gerar os embeddings textuais (TIEnc) é um salto de qualidade fundamental. Essa abordagem, alinhada ao paper do SINKT, cria representações vetoriais ricas em semântica para conceitos e questões, o que potencializa a capacidade do modelo de generalizar e entender as relações de conteúdo.

- **O Poder da Estrutura do Grafo (SIEnc):** Apenas criar um grafo de pré-requisitos não é suficiente; é preciso processá-lo para que o modelo extraia valor dessa estrutura. As implementações de **Caio Wanderly** e **Lucas Parteka** com **GAT (Graph Attention Network)** para o SIEnc foram um grande insight. A GAT permite que o modelo aprenda dinamicamente a importância de cada relação no grafo (conceito-conceito, conceito-questão), refinando os embeddings de uma maneira que uma simples agregação não conseguiria.

- **Modularidade e Boas Práticas de Código:** Os notebooks de **Matheus Lacerda** e **Caio Wanderly** se destacaram pela excelente organização e modularidade, encapsulando cada componente do pipeline (Dataset, TIEnc, SIEnc, GRU) em classes distintas. Isso não apenas torna o código mais legível e profissional, mas também facilita a manutenção, o reuso e a depuração. Adotar essa estrutura é uma melhoria clara para o meu próprio código.

- **Simulação Realista de Dados:** A criação do dataset é uma etapa crítica. **Héber Júnior** e **Eduardo Prasniewski** mostraram abordagens sofisticadas para simular o comportamento dos alunos, introduzindo conceitos como habilidade inata, dificuldade progressiva e fatores

de aprendizado/esquecimento. Isso gera um histórico de interações mais próximo da realidade, resultando em um treinamento de modelo mais robusto.

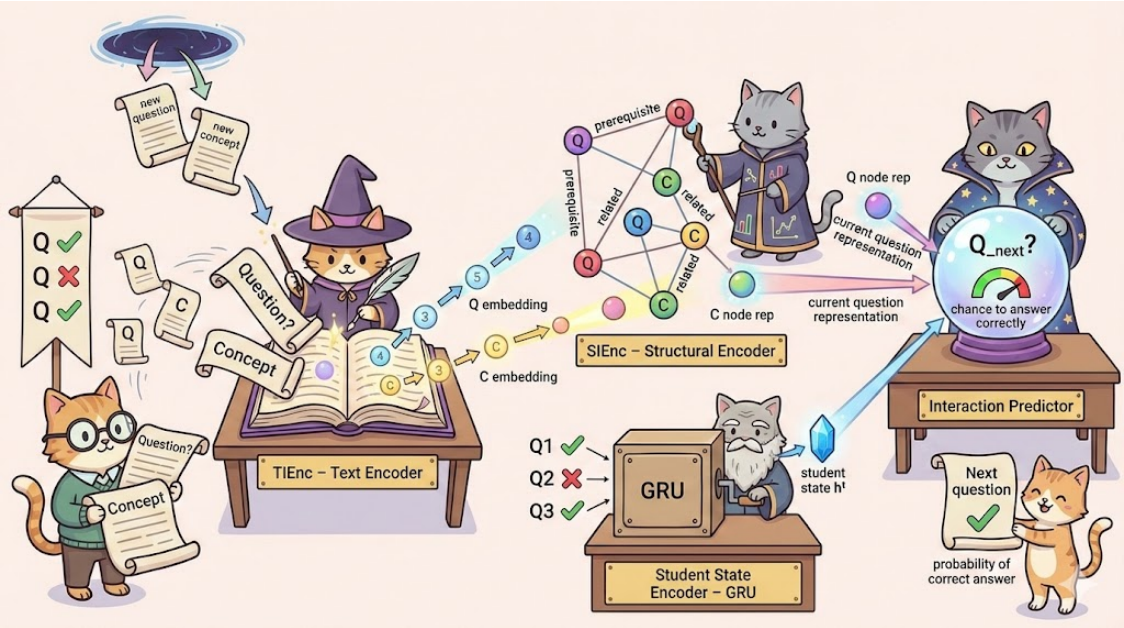

Figure 1: Arquitetura do SINKT

1.2 O que ficou mais claro sobre o pipeline

A revisão cruzada solidificou meu entendimento do pipeline completo, especialmente sobre como os componentes se conectam e a importância de cada um:

- **Dataset → Grafo:** O dataset não serve apenas para o treinamento da GRU, mas é a fonte primária para a construção do grafo heterogêneo, que conecta alunos, questões e conceitos.
- **Grafo → Embeddings (SIEnc):** O propósito do SIEnc ficou muito mais claro: ele não substitui os embeddings textuais (TIEnc), mas os **enriquece**. O processo é: 1) Gerar embeddings semânticos com BERT (TIEnc); 2) Usar a GAT (SIEnc) para agregar informações estruturais dos vizinhos no grafo a esses embeddings, criando uma representação final que une semântica e estrutura.
- **Embeddings → GRU:** A GRU não recebe apenas um ID da questão, mas sim o **embedding enriquecido** da interação (que combina o embedding da questão e o resultado do acerto/erro). Isso permite que o estado oculto da GRU (que representa o conhecimento do aluno) seja atualizado com base em informações muito mais ricas.
- **GRU → Predição:** A predição para uma nova questão q_{t+1} é feita combinando o estado de conhecimento atual do aluno (saída da GRU) com o embedding da nova questão. A interação entre esses dois vetores determina a probabilidade de acerto.

O pipeline, portanto, é uma cascata onde cada etapa refina e agrega valor à informação que será utilizada pela etapa seguinte.

1.3 Erros, Riscos ou Inconsistências Encontrados

Durante a análise, identifiquei alguns riscos e pontos de atenção comuns:

- **Risco de Embeddings Simplistas:** Notebooks que utilizam embeddings aleatórios (como o meu, inicialmente) correm o risco de ter um modelo com baixo desempenho, pois falta a informação semântica necessária para que a GRU e a GAT operem de forma eficaz.
- **Subutilização do Grafo:** Alguns notebooks constroem o grafo, mas não o utilizam efetivamente para enriquecer os embeddings (ausência de uma GNN como GAT ou GCN). Nesses casos, o grafo serve apenas para visualização, o que é uma subutilização do seu potencial.
- **Complexidade da Geração do Dataset:** A lógica para gerar um dataset sintético pode se tornar muito complexa e, se não for bem documentada, pode ser difícil de entender e replicar. É crucial equilibrar realismo com simplicidade.
- **Falta de Validação Cruzada:** A maioria dos notebooks (incluindo o meu) utiliza uma simples divisão treino/teste. Para um resultado mais robusto e para avaliar a estabilidade do modelo, a implementação de validação cruzada (como K-Fold) seria uma melhoria importante, como sugerido no meu plano de ação.

1.4 Dúvidas que Surgiram

- Como o SINKT lida com o problema do "cold start" para **novos alunos** (e não apenas novas questões)? O paper foca na natureza indutiva para questões, mas a abordagem para alunos com zero histórico de interações ainda não está totalmente clara para mim.
- Em um cenário de produção, com que frequência o **grafo de pré-requisitos** deveria ser atualizado pela LLM? Seria um processo em batch periódico ou uma atualização em tempo real a cada novo conceito adicionado?

1.5 Eleição do Melhor Código da FASE 2

Meu voto: Matheus Lacerda

Justificativa:

O notebook de Matheus Lacerda foi, em minha avaliação, o mais completo e bem executado. Ele implementou todos os 7 componentes do pipeline de forma clara e funcional, com destaque para a **integração com a LLM para geração dinâmica do grafo**, que era um dos pontos mais desafiadores. O código é altamente modular, com uso consistente de classes e funções, e a presença de tratamento de erros e logging demonstra uma maturidade técnica e uma preocupação com a robustez que o diferencia dos demais. Embora não tenha implementado GAT ou BERT, sua estrutura de código é a mais sólida e preparada para receber essas melhorias.

2 Relatório de Progresso

2.1 Sobre os 7 Componentes do Pipeline SINKT

O pipeline SINKT é composto por 7 componentes essenciais que trabalham em conjunto para realizar o Knowledge Tracing (rastreamento de conhecimento). A seguir, uma breve explicação de cada um:

2.1.1 1. Dataset Creation (Criação do Dataset)

Geração ou carregamento de dados de interações entre alunos e questões. Cada interação contém informações sobre o aluno, a questão respondida, o conceito testado, se houve acerto ou erro, e o timestamp da interação.

2.1.2 2. Graph Construction (Construção do Grafo)

Criação do grafo de conhecimento que representa as relações de pré-requisito entre conceitos. Este grafo pode ser estático (definido manualmente) ou gerado dinamicamente usando LLMs. O grafo também pode incluir relações entre conceitos e questões.

2.1.3 3. Embeddings (Geração de Embeddings)

Criação de representações vetoriais (embeddings) para conceitos e questões. Esses vetores capturam informações semânticas e são a base para o processamento neural. Podem ser gerados de forma aleatória, usando Word2Vec, BERT, ou outros métodos.

2.1.4 4. GRU Model (Modelo GRU)

Implementação da Gated Recurrent Unit, uma rede neural recorrente que rastreia o estado de conhecimento do aluno ao longo do tempo. A GRU processa sequencialmente as interações do aluno, atualizando seu estado interno a cada nova questão respondida.

2.1.5 5. Training Loop (Loop de Treinamento)

Processo iterativo de treinamento do modelo, incluindo forward pass, cálculo de loss (função de perda), backpropagation e otimização dos pesos. O modelo aprende a prever acertos/erros com base nas interações históricas.

2.1.6 6. Prediction (Predição)

Capacidade do modelo treinado de fazer predições sobre o desempenho futuro do aluno em questões específicas. Esta é a funcionalidade principal do Knowledge Tracing: prever se o aluno acertará ou errará a próxima questão.

2.1.7 7. Visualization (Visualização)

Geração de gráficos e visualizações para análise dos resultados, incluindo o grafo de conceitos, curvas de loss/acurácia durante o treinamento, e análise comparativa entre predições e resultados reais.

2.2 Melhorias Identificadas para Implementação

Após análise do código atual e comparação com as melhores práticas observadas nos notebooks dos colegas, identifiquei as seguintes melhorias que precisam ser implementadas no meu notebook:

2.2.1 Integração Completa com IA no Fluxo do Grafo

Atualmente, o grafo de pré-requisitos é gerado de forma estática. É necessário integrar a IA (Gemini) de forma mais robusta no pipeline para:

- Gerar dinamicamente as relações de pré-requisito entre conceitos usando prompts estruturados
- Implementar cache das respostas da IA para evitar chamadas repetidas
- Validar a qualidade das relações geradas pela IA

2.2.2 Implementação Formal de TIEnc (Textual Information Encoder)

O TIEnc atual usa embeddings aleatórios. Preciso implementar:

- **BERT** (bert-base-uncased) para encoding textual de conceitos e questões
- Congelamento dos pesos do BERT (como especificado no paper SINKT)
- Estratégia de pooling ([CLS] token ou média dos tokens)
- Classe `TextualInformationEncoder` com método `encode_texts(lista_de_textos) -> tensor`

2.2.3 Implementação Formal de SIEnc (Structural Information Encoder)

O SIEnc atual é simplificado. Preciso implementar:

- **GAT (Graph Attention Network)** para grafo heterogêneo (conceitos + questões)
- Agregação multi-relacional:
  - Conceito $\rightarrow$ Conceito (pré-requisitos)
  - Conceito $\rightarrow$ Questão (questões que testam o conceito)
  - Questão $\rightarrow$ Conceito (conceitos testados pela questão)
- Jumping Knowledge (preservação do embedding textual inicial)
- 1-2 camadas de GAT (k=2 conforme paper)

2.2.4 Validação Cruzada K-Fold

Atualmente uso apenas divisão treino/teste simples. Preciso implementar:

- Validação cruzada 5-fold para avaliação mais robusta
- Cálculo de métricas médias e desvio padrão
- Análise de estabilidade do modelo

2.3 Análise de Notebooks

2.3.1 Matheus Lacerda

Resumo
Implementação completa do pipeline SINKT (7/7 componentes) com destaque para a integração com LLM na geração do grafo de pré-requisitos. O código é altamente modular, utilizando classes e funções com type hints consistentes. Demonstra excelente compreensão do fluxo completo: Dataset $\rightarrow$ Grafo $\rightarrow$ Embeddings $\rightarrow$ GRU $\rightarrow$ Predição. A implementação de tratamento de erros e logging do progresso do treinamento são pontos fortes significativos.

Pontos Fortes

- Dataset bem estruturado
- Grafo conceitual implementado corretamente
- GRU implementada seguindo a arquitetura do SINKT
- Código altamente modular com classes e funções
- Type hints consistentes em todo o código
- **Integração com LLM** para geração dinâmica de grafo
- Define seed para reprodutibilidade
- Implementa tratamento de erros

O que posso aplicar no meu notebook

- **Integração com LLM:** Implementar geração dinâmica do grafo usando Gemini com cache e fallback
- **Tratamento de erros:** Adicionar try-except em chamadas de API e operações críticas
- **Logging estruturado:** Registrar progresso do treinamento com métricas por época
- **Modularização:** Criar classes separadas para cada componente (TIEnc, SIEnc, GRU)

2.3.2 Caio Wanderly

Resumo
Implementação técnica sólida do pipeline SINKT (7/7 componentes) com destaque para a implementação de GAT (Graph Attention Network) para o SIEnc. O código é direto e funcional, focado na implementação técnica sem células de Markdown explicativas. Demonstra profundo conhecimento de Graph Neural Networks e sua aplicação em Knowledge Tracing. A modularização com classes é bem executada.

Pontos Fortes

- Dataset bem estruturado
- Grafo conceitual implementado corretamente
- GRU implementada seguindo a arquitetura do SINKT
- Código altamente modular com classes e funções
- Type hints consistentes em todo o código
- **Implementa GAT para SIEnc** (Structural Information Encoder)
- Define seed para reprodutibilidade
- Registra progresso do treinamento

O que posso aplicar no meu notebook

- **GAT para SIEnc:** Implementar Graph Attention Network para enriquecer embeddings com informação estrutural
- **Grafo heterogêneo:** Modelar explicitamente diferentes tipos de nós (conceitos vs questões) e arestas
- **Agregação multi-relacional:** Implementar agregação separada para C→C, C→Q e Q→C

2.3.3 Lucas Parteka

Resumo
O único a implementar tanto BERT para TIEnc quanto GAT para SIEnc. Embora o pipeline esteja com 6/7 componentes (predições incompletas), a implementação do BERT e GAT parece coerente para os testes Faltam docstrings em algumas funções, mas a estrutura geral é sólida.

Pontos Fortes

- Dataset bem estruturado
- Grafo conceitual implementado corretamente
- GRU implementada seguindo a arquitetura do SINKT
- Código altamente modular com classes e funções

- **Implementa BERT para TIEnc** (Textual Information Encoder)
- **Implementa GAT para SIEnc** (Structural Information Encoder)
- Define seed para reprodutibilidade
- Registra progresso do treinamento

O que posso aplicar no meu notebook

- **BERT para TIEnc:** Implementar encoding textual usando transformers (bert-base-uncased)
- **Congelamento de pesos:** Congelar BERT e usar apenas como feature extractor
- **GAT para SIEnc:** Implementar attention mechanism no grafo heterogêneo
- **Jumping Knowledge:** Preservar embedding textual inicial através das camadas do GAT

2.3.4 Lucas Senzaki

Resumo
Implementação funcional e completa do pipeline SINKT (7/7 componentes). O código é bem organizado com uso de classes e funções, parece demonstrar boa compreensão do modelo. Faltam docstrings em algumas funções. A implementação é sólida e serve como boa referência para um pipeline completo.

Pontos Fortes

- Dataset bem estruturado
- Grafo conceitual implementado corretamente
- GRU implementada seguindo a arquitetura do SINKT
- Código altamente modular com classes e funções
- Define seed para reprodutibilidade
- Registra progresso do treinamento

O que posso aplicar no meu notebook

- **Predições completas:** Implementar seção dedicada para predições com análise detalhada
- **Visualizações:** Adicionar gráficos de análise de predições vs realidade

2.3.5 Pedro Augusto

Resumo
Implementação clara e direta do pipeline SINKT (7/7 componentes), focada em criar um exemplo mínimo e funcional. O código é bem comentado e fácil de seguir, ideal para quem está começando a entender o SINKT. A criação do dataset é simples, mas eficaz para o propósito do experimento.

Pontos Fortes

- Código muito bem comentado e explicado.
- Implementação completa do pipeline, do dataset à predição.
- Define seed para reprodutibilidade.
- Visualização do grafo de conceitos.

O que posso aplicar no meu notebook

- Melhorar os comentários e explicações do meu código.
- Adicionar uma seção de visualização de predições, como a plotagem de probabilidades.

2.3.6 Lucas Timoteo

Resumo
Implementação completa do pipeline SINKT (7/7 componentes) com uma abordagem muito clara e bem documentada. O código é organizado em seções que correspondem diretamente às etapas do pipeline, facilitando o entendimento. A geração do dataset e a implementação da GRU são sólidas.

Pontos Fortes

- Código bem organizado e documentado com células de Markdown.
- Implementação completa e funcional do pipeline.
- Define seed para reprodutibilidade.
- Visualização do grafo e das predições.

O que posso aplicar no meu notebook

- Estruturar o notebook em seções claras, com explicações em Markdown para cada etapa.
- Adicionar uma análise mais detalhada das predições, comparando o previsto com o real.

2.3.7 Héber Júnior

Resumo
Implementação completa e robusta do pipeline SINKT (7/7 componentes), com uma abordagem detalhada na criação do dataset, simulando habilidades de alunos e dificuldades de conceitos de forma progressiva. O código é bem estruturado e inclui métricas de avaliação detalhadas.

Pontos Fortes

- Dataset artificial bem elaborado, com progressão de dificuldade.
- Implementação completa do pipeline.
- Avaliação do modelo com múltiplas métricas (acurácia, precisão, recall, F1-score).
- Código bem organizado e com funções claras.

O que posso aplicar no meu notebook

- Aprimorar a geração do dataset para simular um ambiente de aprendizado mais realista.
- Incluir uma avaliação mais completa do modelo, com diferentes métricas.

2.3.8 Eduardo Prasniewski

Resumo
Implementação focada na GRU e na visualização do conhecimento do aluno. O pipeline está parcialmente completo (6/7), mas a parte de visualização é um destaque, mostrando o domínio do aluno sobre os conceitos de forma gráfica e intuitiva. A lógica para a criação do dataset também é interessante, utilizando uma função logística para o fator de pré-requisitos.

Pontos Fortes

- Excelente visualização do estado de conhecimento do aluno.
- Uso de função logística para uma transição mais suave no aprendizado.
- Código bem comentado e com boas explicações.

O que posso aplicar no meu notebook

- Criar uma visualização do estado de conhecimento do aluno, mostrando a evolução do domínio dos conceitos.
- Experimentar diferentes funções de ativação e transição para modelar o aprendizado.

2.4 Sugestões de Melhoria

Esta seção servirá como uma autoavaliação crítica, comparando-a com as implementações dos colegas para identificar pontos de melhoria, conforme solicitado.

2.4.1 Resumo da Minha Implementação (Erick)

Minha implementação buscou criar um pipeline SINKT completo (7/7 componentes) utilizando um dataset realista com conceitos e questões de programação em Python. Um dos focos foi a integração com o Gemini para gerar dinamicamente o grafo de pré-requisitos, além de simular perfis de alunos para criar um histórico de interações mais autêntico. O código é estruturado, comentado e inclui visualizações.

2.4.2 Pontos Fortes da Minha Implementação

- **Dataset Realista:** Utiliza conceitos e questões reais de Python, o que torna o problema mais tangível.
- **Integração com LLM:** Gera o grafo de pré-requisitos dinamicamente usando o Gemini, com prompts estruturados.
- **Simulação de Alunos:** Cria perfis de alunos (iniciante, intermediário, avançado) para gerar dados de interação mais realistas.
- **Pipeline Completo:** Implementa todas as 7 fases do pipeline, da criação do dataset à predição e visualização.

2.4.3 Pontos a Melhorar com Base na Análise dos Colegas

Após revisar os notebooks dos meus colegas, identifiquei várias oportunidades claras para aprimorar minha própria implementação, alinhando-a com as melhores práticas observadas.

1. **Implementar BERT para TIEnc (Textual Information Encoder)**
    - **Observação:** Meu notebook usa embeddings aleatórios, que são semanticamente pobres. **Lucas Parteka** implementou o **BERT** para gerar embeddings textuais, o que é uma abordagem muito mais robusta e alinhada ao paper do SINKT.
    - **Ação para Melhoria:** Substituir os embeddings aleatórios pela classe `TextualInformationEncoder` utilizando o `bert-base-uncased`. Os pesos do BERT devem ser congelados, usando-o apenas como um extrator de características para gerar vetores ricos em semântica para os conceitos e questões.
2. **Implementar GAT para SIEnc (Structural Information Encoder)**
    - **Observação:** Embora meu notebook gere um grafo, ele não é processado por uma GNN (Graph Neural Network) para enriquecer os embeddings. **Caio Wanderly** e **Lucas Parteka** implementaram uma **GAT (Graph Attention Network)** para o SIEnc, o que permite ao modelo aprender a importância das relações estruturais.
    - **Ação para Melhoria:** Implementar a classe `StructuralInformationEncoder` usando uma GAT. O grafo deve ser heterogêneo (nós de conceito e questão) e a GAT deve aprender a agregar informações dos vizinhos (outros conceitos e questões) para refinar os embeddings textuais iniciais.

3. **Expandir a Avaliação do Modelo**

    - **Observação: Héber Júnior** utilizou um conjunto mais completo de métricas de avaliação, incluindo precisão, recall e F1-score, além da acurácia.
    - **Ação para Melhoria:** Adicionar um relatório de classificação mais completo ao final do treinamento e predição. Isso fornecerá uma visão mais detalhada do desempenho do modelo, especialmente em casos de desbalanceamento de classes (mais acertos que erros, ou vice-versa).

4. **Melhorar as Visualizações**

    - **Observação: Eduardo Prasniewski** criou uma excelente visualização que mostra o estado de conhecimento de um aluno em um grafo, com cores indicando o nível de domínio.
    - **Ação para Melhoria:** Criar uma nova seção de visualização para plotar o estado de conhecimento de um aluno específico, mostrando como o domínio de cada conceito evolui ao longo do tempo. Isso tornaria a interpretação do modelo muito mais intuitiva.